APPENDIX

```
;-----------------------------------------------------------------------
;
;       PRO7000 DC Motor Operator
;       Manual forces, automatic limits
;       New learn switch for learning the limits
;
;       Code based on Flex GDO
;
;-----------------------------------------------------------------------
;
;-----------------------------------------------------------------------
;       Notes:
;-----------------------------------------------------------------------
;  -- Motor is controlled via two Form C relays to control direction
;  -- Motor speed is controlled via a fet (2 IRF540's in parallel) with a
;       phase control PWM applies.
;  -- Wall control (and RS232) are P98 with a redundant smart button and
;       command button on the logic board
;-----------------------------------------------------------------------
;
;---------------------------------------------------------------------------
;       Flex GDO Logic Board
;       Fixed AND Rolling Code Functionality
;       Learn from keyless entry transmitter
;       Posi-lock
;       Turn on light from broken IR beam (when at up limit)
;       Keyless entry temporary password based on number of hours or number
;       of activations.  (Rolling code mode only)
;
;       GDO is initialized to a 'clean slate' mode when the memory is erased.
;       In this mode, the GDO will receive either fixed or rolling codes.
;       When the first radio code is learned, the GDO locks itself into that
;       mode (fixed or rolling) until the memory is again erased.
;
;       Rolling code derived from the Leaded67 code
;       Using the 8K zilog 233 chip
;       Timer interrupt needed to be 2X faster
;
;---------------------------------------------------------------------------
;       Revision History
;---------------------------------------------------------------------------
;       Revision 1.1:
;       -- Changed light from broken IR beam to work in both fixed and rolling
;          modes.
;       -- Changed light from IR beam to work only on beam break, not on beam
;          block.
;
;       Revision 1.2:
;       -- Learning rolling code formerly erased fixed code.  Mode is now
;          determined by first transmitter learned after radio erase.
;
;       Revision 1.3:
;       -- Moved radio interrupt disable to reception of 20 bits.
;       -- Changed mode of radio switching.  Formerly toggled upon radio error,
;          now switches in pseudo-random fashion depending upon value of
;          125 ms timer.
;
;       Revision 1.4:
;       -- Optimized portion of radio after bit value is determined.  Used
;          relative addressing to speed code and minimize ROM size.
;
;       Revision 1.5:
;       -- Changed mode of learning transmitters. Learn command is now
;          light-command, learn light is now light-lock, and learn open/close/
;          stop is lock-command.   (Command was press light, press command,
;          release light, release command, worklight was press light, press command,
;          release command, release light, o/c/s was press lock, press command,
;          release command, release lock.  This caused DOG2 to reset)
;
```

```
;       Revision 1.6:
;       -- Light button and light transmitter now ignored during travel.
;          Switch data cleared only after a command switch is checked.
;
;       Revision 1.7:
;       -- Rejected fixed mode (and fixed mode test) when learning light and
;          open/close/stop transmitters.
;
;       Revision 1.8:
;       -- Changed learn from wall control to work only when both switches are
;          held.  Modified force pot. read routine (moved enabling of blank
;          time and disabling of interrupts).  Fixed mode now learns command
;          with any combination of wall control switches.
;
;       Revision 1.9:
;       -- Changed PWM output to go from 0-50% duty cycle.  This eliminated the
;          problem of PWM interrupts causing problems near 100% duty cycle.
;          THIS REVISION REQUIRES A HARDWARE CHANGE.
;
;       Revision 1.9A:
;       -- Enabled ROM checksum.  Cleaned up documentation.
;
;       Revision 2.0:
;       -- Blank time noise immunitity.  If noise signal is detected during blank time the data
;          already recieved is not thrown out.  The data is retained, and the noise
;          pulse is identified as such.  The interrupt is enabled to contine to look
;          for the sync pulse.
;
;       Revision 2.0A:
;       -- On the event that the noise pulse is of the same duration as the sync pulse,
;          the time between sync and first data pulse (inactive time) is measured.  The
;          inactive time is 5.14ms for billion code and 2.4ms for rolling code.  If it is
;          determined that the previously received sync is indeed a noise pulse, the pulse
;          is thrown out and the micro continuies to look for a sync pulse as in Rev. 2.0.
;
;       Revision 2.1:
;       -- To make the blank time more impervious to noise, the sync pulses are
;          differentiated between.  Fixed max width is 4.6ms, roll max width is 2.3ms.
;          This is simular to the inactive time check done in Rev.2.0A.
;
;       Revision 2.2:
;       -- The worklight function; when the IR beam is broken and the door is at the up limit
;          the light will turn on for 4.5 min.  This revision allows the worklight function to
;          be enabled and disabled by the user.  The function will come enabled from the factory.
;          To disable, with the light off press and hold the light button for 7 sec.  The light will
;          come on and after 7 sec. the function is disabled the light will turn off.  To enable the
;          function, turn the light on, release the button, then press and hold the light button
;          down for 7 sec.  The light will turn off and after the function has been enable in 7 sec.
;          the light will turn on.
;
;       Revision 3.0:
;       -- Integrated in functionality for Siminor rolling code transmitter.  The Siminor
;          transmitter may be received whenever a C code transmitter may be received.
;          Siminor transmitters are able to perform as a standard command or as a light
;          control transmitter, but not as an open/close/stop transmitter.
;
;       Revision 3.1:
;       -- Modified handling of rolling code counter (in mirroring and adding) to improve
;          efficiency and hopefully kill all short cycles when a radio is jammed on the
;          air.
;-------------------------------
;       PRO7000
;-------------------------------
;       Revision 0.1:
;       -- Removed physical limit tests
;       -- Disabled radio temporarily
;       -- Put in sign bit test for limits
;       -- Automatic limits working
;
```

```
;       Revision 0.2:
;       -- Provided for traveling up when too close to limit
;
;       Revision 0.3:
;       -- Changed force pot. read to new routine.
;       -- Disabled T1 interrupt and all old force pot. code
;       -- Disabled all RS232 output
;
;       Revision 0.4:
;       -- Added in (veerrrry) rough force into pot. read routine
;
;       Revision 0.5:
;       -- Changed EEPROM in comments to add in up limit, last operation, and
;          down limit.
;       -- Created OnePass register
;       -- Added in limit read from nonvolatile when going to a moving state
;       -- Added in limit read on power-up
;       -- Created passcounter register to keep track of pass point(s)
;       -- Installed basic wake-up routine to restore position based on last state
;
;       Revision 0.6:
;       -- Changed RPM time read to routine used in P98 to save RAM
;       -- Changed operation of RPM forced up travel
;       -- Implemented pass point for one-pass-point travel
;
;       Revision 0.7:
;       -- Changed pass point from single to multiple (no EEPROM support)
;
;       Revision 0.8:
;       -- Changed all SKIPRADIO loads from 0xFF to NOEECOMM
;       -- Installed EEPROM support for multiple pass points
;
;       Revision 0.9:
;       -- Changed state machine to handle wake-up (i.e. always head towards
;          the lowest pass point to re-orient the GDO)
;
;       Revision 0.10:
;       -- Changed the AC line input routine to work off full-wave rectified
;          AC coming in
;
;       Revision 0.11:
;       -- Installed the phase control for motor speed control
;
;       Revision 0.12:
;       -- Installed traveling down if too near up limit
;       -- Installed speed-up when starting travel
;       -- Installed slow-down when ending travel
;
;       Revision 0.13:
;       -- Re-activated the C code
;
;       Revision 0.14:
;       -- Added in conditional assembly for Siminor radio codes
;
;       Revision 0.15:
;       -- Disabled old wall control code
;       -- Changed all pins to conform with new layout
;       -- Removed unused constants
;       -- Commented out old wall control routine
;       -- Changed code to run at 6MHz
;
;       Revision 0.16
;       -- Fixed bugs in Flex radio
;
;       Revision 0.17
;       -- Re-enabled old wall control.  Changed command charging time to 12 ms
;          to fix FMEA problems with IR protectors.
;
;       Revision 0.18
;       -- Turned on learn switch connected to EEPROM clock line
;
;       Revision 0.19
;       -- Eliminated unused registers
;       -- Moved new registers out of radio group
;       -- Re-enabled radio interrupt
;
;       Revision 0.20
;       -- Changed limit test to account for "lost" position
;       -- Re-wrote pass point routine
;
;       Revision 0.21
;       -- Changed limit tests in state setting routines
;       -- Changed criteria for looking for lost position
;       -- Changed lost operation to stop until position is known
;
;       Revision 0.22:
;       -- Added in L_A_C state machine to learn the limits
;               -- Installed learn-command to go into LAC mode
;               -- Added in command button and learn button jog commands
;               -- Disabled limit testing when in learn mode
;               -- Added in LED flashing for in learn mode
;               -- Added in EVERYTHING with respect to learning limits
;       -- NOTE:  LAC still isn't working properly!!!
;
;       Revision 0.23:
;       -- Added in RS232 functionality over wall control lines
;
;       Revision 0.24:
;       -- Touched up RS232 over wall control routine
;       -- Removed 50Hz force table
;       -- Added in fixes to LAC state machine
;
;       Revision 0.25:
;       -- Added switch set and release for wall control (NOT smart switch)
;          into RS232 commands (Turned debouncer set and release in to subs)
;       -- Added smart switch into RS232 commands (smart switch is also a sub)
;       -- Re-enabled pass point test in ':' RS232 command
;       -- Disabled smart switch scan when in RS232 mode
;       -- Corrected relative references in debouncer subroutines
;       -- RS232 'F' command still needs to be fixed
;
;       Revision 0.26:
;       -- Added in max. force operation until motor ramp-up is done
;       -- Added in clearing of slowdown flag in set_any routine
;       -- Changed RPM timeout from 30 to 60 ms
;
;       Revision 0.27:
;       -- Switched phase control to off, then on (was on, then off) inside
;          each half cycle of the AC line (for noise reduction)
;       -- Changed from 40ms unit max. period to 32 (will need further changes)
;       -- Fixed bug in force ignore during ramp (previously jumped from down to
;          up state machine!)
;       -- Added in complete force ignore at very slow part of ramp (need to change
;          this to ignore when very close to limit)
;       -- Removed that again
;       -- Bug fix -- changed force skip during ramp-up.  Before, it kept counting
;          down the force ignore timer.
;
;       Revision 0.28:
;       -- Modified the wall control documentation
;       -- Installed blinking the wall control on an IR reversal instead of the
;          worklight
;       -- Installed blinking the wall control when a pass point is seen
;
;       Revision 0.29:
;       -- Changed max. RPM timeout to 100 ms
;       -- Fixed wall control blink bug
;       -- Raised minimum speed setting
;       NOTE:  Forces still need to be set to accurate levels
;
;       Revision 0.30:
;       -- Removed 'ei' before setting of pcon register
;       -- Bypassed slow-down to limit during learn mode
;
;       Revision 0.31:
;       -- Changed force ramp to a linear FORCE ramp, not a linear time ramp
;          -- Installed a look-up table to make the ramp more linear.
;       -- Disabled interrupts during radio pointer match
;       -- Changed slowdown flag to a up-down-stop ramping flag
;
;       Revision 0.32:
;       -- Changed down limit to drive lightly into floor
;       -- Changed down limit when learning to back off of floor a few pulses
;
;       Revision 0.33:
;       -- Changed max. speed to 2/3 when a short door is detected
;
;       Revision 0.34:
;       -- Changed light timer to 2.5 minutes for a 50 Hz line, 4.5 minutes for
;          a 60 Hz line.  Currently, the light timer is 4.5 minutes WHEN THE UNIT
;          FIRST POWERS UP.
;       -- Fixed problem with leaving RF set to an extended group
;
;       Revision 0.35:
;       -- Changed starting position of pass point counter to 0x30
;
;       Revision 0.36:
;       -- Changed algorithm for finding down limit to cure stopping at the floor
;          during the learn cycle
;       -- Fixed bug in learning limits:  Up limit was being updated from EEPROM
;          during the learn cycle!
;       -- Changed method of checking when limit is reached:  calculation for
;          distance to limit is now ALWAYS performed
;       -- Added in skipping of limit test when position is lost
;
;       Revision 0.37:
;       -- Revised minimum travel distance and short door constants to reflect
;          approximately 10 RPM pulses / inch
;
;       Revision 0.38:
;       -- Moved slowstart number closer to the limit.
;       -- Changed backoff number from 10 to 8
;
;       Revision 0.39:
;       -- Changed backoff number from 8 to 12
;
;       Revision 0.40:
;       -- Changed task switcher to unburden processor
;       -- Consolidated tasks 0 and 4
;       -- Took extra unused code out of tasks 1, 3, 5, 7
;       -- Moved aux light and 4 ms timer into task 6
;       -- Put state machine into task 2 only
;       -- Adjusted auto_delay, motdel, rpm_time_out, force_ignore, motor_timer,
;          obs_count for new state machine tick
;       -- Removed force_pre prescaler (no longer needed with 4ms state machine)
;       -- Moved updating of obs_count to one ms timer for accuracy
;       -- Changed autoreverse delay timer into a byte-wide timer because it was
;          only storing an 8 bit number anyways...
;       -- Changed flash delay and light timer constants to adjust for 4ms tick
;
;       Revision 0.41
;       -- Switched back to 4MHz operation to account for the fact that Zilog's
;          Z86733 OTP won't run at 6MHz reliably
;
;       Revision 0.42:
;       -- Extended RPM timer so that it could measure from 0 - 524 ms with
;          a resolution of 8us
```

```
;
;       Revision 0.43:
;       -- Put in the new look-up table for the force pots (max RPM pulse period
;          multiplied by 20 to scale it for the various speeds).
;       -- Removed taskswitch because it was a redundant r gister
;       -- Removed extra call to the auxlight routine
;       -- Removed register 'temp' because, as far as I can tell, it does nothing
;       -- Removed light_pre register
;       -- Eliminated 'phase' register because it was never used
;       -- Put in preliminary divide for scaling the force and speed
;       -- Created speedlevel AND IDEAL speed registers, which are not yet used
;
;       Revision 0.47:
;       -- Undid the work of revisions 0.44 through 0.46
;       -- Changed ramp-up and ramp-down to an adaptive ramp system
;       -- Changed force compare from subtract to a compare
;       -- Removed force ignore during ramp (was a kludge)
;       -- Changed max. RPM time out to 500 ms static
;       -- Put WDT kick in just before main loop
;       -- Fixed the word-wise T0EXT register
;       -- Set default RPM to max. to fix problem of not ramping up
;
;       Revision 0.48:
;       -- Took out adaptive ramp
;       -- Created look-ahead speed feedback in RPM pulses
;
;       Revision 0.49:
;       -- Removed speed feedback (again)
;          NOTE:  Speed feedback isn't necessarily impossible, but, after all my
;                 efforts, I've concluded that the design time necessary (a large
;                 amount) isn't worth the benefit it gives, especially given the
;                 current time constraints of this project.
;       -- Removed RPM_SET_DIFF lo and hi registers, along with IDEAL_SPEED lo
;          and hi registers (only need them for speed feedback)
;       -- Deleted speedlevel register (no longer needed)
;       -- Separated the start of slowdown for the up and down directions
;       -- Lowered the max. speed for short doors
;       -- Set the learn button to NOT erase the memory when jogging limits
;
;       Revision 0.50:
;       -- Fixed the force pot read to actually return a value of 0-64
;       -- Set the msx. RPM period time out to be equivalent to the force setting
;
;       Revision 0.51:
;       -- Added in P2M_SHADOW register to make the following possible:
;       -- Added in flashing warning light (with auto-detect)
;
;       Revision 0.52:
;       -- Fixed the variable worklight timer to have the correct value on
;          power-up
;       -- Re-enabled the reason register and stackreason
;       -- Enabled up limit to back off by one pulse if it appears to be
;          crashing the up stop bolt.
;       -- Set the door to ignore commands and radio when lost
;       -- Changed start of down ramp to 220
;       -- Changed backoff from 12 to 9
;       -- Changed drive-past of down limit to 9 pulses
;
;       Revision 0.53:
;       -- Fixed RS232 '9' and 'F' commands
;       -- Implemented RS232 'K' command
;       -- Removed 'M', 'P', and 'S' commands
;       -- Set the learn LED to always turn off at the end of the
;          learn limits mode
;
;       Revision 0.54:
;       -- Reversed the direction of the pot. read to correct the direction
;          of the min. and max. forces when dialing the pots.
;       -- Added in "U" command (currently does nothing)
;       -- Added in "V" command to read force pot. values
;
;       Revision 0.55:
;       -- Changed number of pulses added in to down limit from 9 to 16
;
;       Revision 0.56:
;       -- Changed backoff number from 16 back to 9 (not 8!)
;       -- Changed minimum force/speed from 4/20 to 10/20
;
;       Revision 0.57:
;       -- Changed backoff number back to 16 again
;       -- Changed minimum force/speed from 10/20 back to 4/20
;       -- Changed learning speed from 10/20 to 20/20
;
;       Revision 0.58:
;       -- Changed learning speed from 20/20 to 12/20 (same as short door)
;       -- Changed force to max. during ramp-up period
;       -- Changed RPM timeout to a static value of 500 ms
;       -- Changed drive-past of limit from 1" to 2" of trolley travel
;          (Actually, changed the number from 10 pulses to 20 pulses)
;       -- Changed start of ramp-up from 1 to 4 (i.e. the power level)
;       -- Changed the algorithm when near the limit -- the door will no
;          longer avoid going toward the limit, even if it is too close
;
;       Revision 0.59:
;       -- Removed ramp-up bug from autoreverse of GDO
;
;       Revision 0.60:
;       -- Added in check for pass point counter of -1 to find position when lost
;       -- Change in waking up when lost.  GDO now heads toward pass point only on
;          first operation after a power outage.  Heads down on all subsequent
;          operations.
;       -- Created the "limits unknown" fault and prevented the GDO from traveling
;          when the limits are not set at a reasonable value
;       -- Cleared the fault code on entering learn limits mode
;       -- Implemented RS232 'H' command
;
;       Revision 0.61:
;       -- Changed limit test to look for trolley exactly at the limit position
;       -- Changed search for pass point to erase limit memory
;       -- Changed setup position to 2" above the pass point
;       -- Set the learn LED to turn off whenever the L_A_C is cleared
;       -- Set the learn limits mode to shut off whenever the worklight times out
;
;       Revision 0.62:
;       -- Removed test for being exactly at down limit (it disabled the drive into
;          the limit feature)
;       -- Fixed bug causing the GDO to ignore force when it should autoreverse
;       -- Added in ignoring commands when lost and traveling up
;
;       Revision 0.63:
;       -- Installed MinSpeed register to vary minimum speed with force pot
;          setting
;       -- Created main loop routine to scale the min speed based on force pot.
;       -- Changed drive-past of down limit from 20 to 30 pulses (2" to 3")
;
;       Revision 0.64:
;       -- Changed learning algorithm to utilize block.  (Changed autoreverse to
;          add in 1/2" to position instead of backing the trolley off of the floor)
;       -- Enabled ramp-down when nearing the up limit in learn mode
;
;       Revision 0.65:
;       -- Put special case in speed check to enable slow down near the up limit
;
;       Revision 0.66:
;       -- Changed ramp-up:  Ramping up of speed is now constant -- the ramp-down
;          is the only ramp affected by the force pot. setting
;       -- Changed ramp-up and ramp-down tests to ensure that the GDO will get UP
;          to the minimum speed when we are inside the ramp-down zone (The above
;          change nec ssitated this)
;       -- Chang d down limit to add in 0.2" instead of 0.5"
;
;       Revision 0.67:
;       -- Removed minimum travel test in set_arev_state
;       -- Mov d minimum distance of down limit from pass point from 5" to 2"
;       -- Disabled moving pass point when only one pass point has been seen
;
;       Revision 0.68:
;       -- Set error in learn state if no pass point is seen
;
;       Revision 0.69:
;       -- Added in decrement of pass point counter in learn mode to kill bugs
;       -- Fixed bug:  Force pots were being ignored in the learn mode
;       -- Added in filtering of the RPM (RPM_FILTER register and a routine in
;          the one ms timer)
;       -- Added in check of RPM filter inside RPM interrupt
;       -- Added in polling RPM pin inside RPM interrupt
;       -- Re-enabled stopping when in learn mode and position is lost
;
;       Revision 0.70:
;       -- Removed old method of filtering RPM
;       -- Added in a "debouncer" to filter the RPM
;
;       Revision 0.71:
;       -- Changed "debouncer" to automatically vector low whenever an RPM pulse
;          is considered valid
;
;       Revision 0.72:
;       -- Changed number of pulses added in to down limit to 0.  Since the actual
;          down limit test checks for the position to be BEYOND the down limit
;          this is the equivalent of adding one pulse into the down limit
;
;       Revision 0.74:
;       -- Undid the work of rev. 0.73
;       -- Changed number of pulses added in to down limit to 1.  Noting the comment
;          in rev. 0.72, this means that we are adding in 2 pulses
;       -- Changed learning speed to vary between 8/20 and 12/20, depending upon
;          the force pot. setting
;
;       Revision 0.75:
;       -- Installed power-up chip ID on P22, P23, P24, and P25
;          Note:  ID is on P24, P23, and P22.  P25 is a strobe to signal valid data
;                 First chip ID is 001 (with strobe, it's 1001)
;       -- Changed set_any routine to re-enable the wall control just in case we
;          stopped while the wall control was being turned off (to avoid disabling
;          the wall control completely)
;       -- Changed speed during learn mode to be 2/3 speed for first seven seconds,
;          then to slow down to the minimum speed to make the limit learning the same
;          as operation during normal travel.
;
;       Revision 0.76:
;       -- Restored learning to operate only at 60% speed
;
;       Revision 0.77:
;       -- Set unit to reverse off of floor and subtract 1" of travel
;       -- Reverted to learning at 40% - 60% of full speed
;
;       Revision 0.78:
;       -- Changed rampflag to have a constant for running at full speed
;       -- Used the above change to simplify the force ignore routine
;       -- Also used it to change the RPM time out.  The time out is now set equal
;          to the pot setting, except during the ramp up when it is set to 500 ms.
;       -- Changed highest force pot setting to be exactly equal to 500ms.
;
;       Revision 0.79:
;       -- Changed setup routine to reverse off block (yet again).  Added in one pulse.
;
;       Revision 1.0:
```

```
;       -- Enabled RS232 version number return
;       -- Enabled ROM checksum.  Cleaned up documentation
;
;       Revision 1.1:
;       -- Tweaked light times for 8.192 ms prescale instead of 8.0 ms prescale
;       -- Changed compare statement inside setvarlight to 'uge' for consistency
;       -- Changed one-shot low time to 2 ms for power line
;       -- Changed one-shot low time to truly count falling-edge-to-falling-edge
;
;       Revision 1.2:
;       -- Eliminated testing for lost GDO in set_up_dir_state (is already taken
;          care of by set_dn_dir_state)
;       -- Created special time for max. run motor timer in learn mode:  50 seconds
;
;       Revision 1.3:
;       -- Fixed bug in set_any to fix stack imbalance
;       -- Changed short door discrimination point to 78"
;
;       Revision 1.4:
;       -- Changed second 'di' to 'ei' in KnowSimCode
;       -- Changed IR protector to ignore for first 0.5 second of travel
;       -- Changed blinking time constant to take it back to 2 seconds before travel
;       -- Changed blinking code to ALWAYS flash during travel, with pre-travel flash
;          when module is properly detected
;       -- Put in bounds checking on pass point counter to keep it in line
;       -- Changed driving into down limit to consider the system lost if floor not seen
;
;       Revision 1.5:
;       -- Changed blinking of wall control at pass point to be a one-shot timer
;          to correct problems with bad passpoint connections and stopping at pass
;          point to cause wall control ignore.
;
;       Revision 1.6:
;       -- Fixed blinking of wall control when indicating IR protector reversal
;          to give the blink a true 50% duty cycle.
;       -- Changed blinker output to output a constant high instead of pulsing.
;       -- Changed P2S_POR to 1010 (Indicate Siminor unit)
;
;       Revision 1.7:
;       -- Disabled Siminor Radio
;       -- Changed P2S_POR to 1011 (Indicate Lift-Master unit)
;       -- Added in one more conditional assembly point to avoid use of simradio label
;
;       Revision 1.8:
;       -- Re-enabled Siminor Radio
;       -- Changed P2S_POR back to 1010 (Siminor)
;       -- Re-fixed blinking of wall control LED for protector reversal
;       -- Changed blinking of wall control LED for indicating pass point
;       -- Fixed error in calculating highest pass point value
;       -- Fixed error in calculating lowest pass point value
;
;       Revision 1.9:
;       -- Lengthened blink time for indicating pass point
;       -- Installed a max. travel distance when lost
;             -- Removed skipping up limit test when lost
;             -- Reset the position when lost and force reversing
;       -- Installed sample of pass point signal when changing states
;
;       Revision 2.0:
;       -- Moved main loop test for max. travel distance (was causing a memory
;          fault before)
;
;       Revision 2.1:
;       -- Changed limit test to use 11000000b instead of 10000000b to ensure
;          only setting up limit when we're actually close.
;
;       Revision 2.2:
;       -- Changed minimum speed scaling to move it further down the pot. rotation.
;          Formula is now:  ((force - 24) / 4) + 4, truncated to 12
;       -- Changed max. travel test to be inside motor state machine.  Max. travel
;          test calculates for limit position differently when the system is lost.
;       -- Reverted limit test to use 10000000b
;       -- Changed some jp's to jr's to conserve code space
;       -- Changed loading of reason byte with 0 to clearing of reason byte (very
;          desperate for space)
;
;       Revision 2.3:
;       -- Disabled Siminor Radic
;       -- Changed P2S_POR to 1011 (Lift-Master)
;
;       Revision 2.4:
;       -- Re-enabled Siminor Radio
;       -- Changed P2S_POR to 1010 (Siminor)
;       -- Changed wall control LED to also flash during learn mode
;       -- Changed reaction to single pass point near floor.  If only one pass point
;          is seen during the learn cycle, and it is too close to the floor, the
;          learn cycle will now fail.
;       -- Removed an ei from the pass point when learning to avoid a race condition
;
;       Revision 2.5:
;       -- Changed backing off of up limit to only occur during learn cycle.  Backs
;          off by 1/2" if learn cycle force stops within 1/2" of stop bolt.
;       -- Removed considering system lost if floor not seen.
;       -- Changed drive-past of down limit to 36 pulses (3")
;       -- Added in clearing of power level whenever motor gets stopped (to turn off
;          the FET's sooner)
;       -- Added in a 40ms delay (using the same MOTDEL register as for the traveling
;          states) to delay the shut-off of the motor relay.  This should enable the
;          motor to discharge some energy before the relay has to break the current
;          flow)
;       -- Created STOPNOFLASH label -- it looks like it should have been there all along
;       -- Moved incrementing MOTDEL timer into head of state machine to conserve space
;
;       Revision 2.6:
;       -- Fixed back-off of up limit to back off in the proper direction
;       -- Added in testing for actual stop state in back-off (before was always backing
;          off the limit)
;       -- Simplified testing for light being on in 'set any' routine; eliminated lights
;          register
;
;       Revision 2.7: (Test-only revision)
;       -- Moved ei when testing for down limit
;       -- Eliminated testing for negative number in radio time calculation
;       -- Installed a primitive debouncer for the pass point (out of paranoia)
;       -- Changed a pass point in the down direction to correspond to a position of 1
;       -- Installed a temporary echo of the RPM signal on the blinker pin
;       -- Temporarily disabled ROM checksum
;       -- Moved three subroutines before address 0101 to save space (2.7B)
;       -- Framed look up using upforce and dnforce registers with di and ei to
;          prevent corruption of upforce or dnforce while doing math (2.7C)
;       -- Fixed error in definition of pot_count register (2.7C)
;       -- Disabled actual number check of RPM perdod for debug (2.7D)
;       -- Added in di at test_up_sw and test_dn_sw for ramping up period(2.7D)
;       -- Set RPM_TIME_OUT to always be loaded to max value for debug (2.7E)
;       -- Set RPM_TIME_OUT to round up by two instead of one (2.7F)
;       -- Removed 2.7E revision (2.7F)
;       -- Fixed RPM_TIME_OUT to round up in both the up and down direction(2.7G)
;       -- Installed constant RS232 output of RPM_TIME_OUT register (2.7H)
;       -- Enabled RS232 'U' and 'V' commands (2.7I)
;       -- Disabled consant output of 2.7H (2.7I)
;       -- Set RS232 'U' to output RPM_TIME_OUT(2.7I)
;       -- Removed disable of actual RPM number check (2.7J)
;       -- Removed pulsing to indicate RPM interrupt (2.7J)
;       -- 2.7J note -- need to remove 'u' command function
;
;       Revision 2.8:
;       -- Removed interrupt enable before resetting rpm_time_out.  This will introduce
;          roughly 30us of extra delay in time measurement, but should take care of
;         nuisance stops.
;      -- Removed push-ing and pop-ing of RP in tasks 2 and 6 to save stack space (2.8B)
;      -- Removed temporary functionality for 'u' command (2.8 Release)
;      -- Re-enabled ROM checksum (2.8 Release)
;
;------------------------------------------------------------------------------
;
;      L_A_C State Machine
;
;
;               73                    77
;            *******                 ****
;    |                *               *
;    |      72       74*          76 *
;    |     Back to     *            *
;   70     Up Lim     ----         ----
;    |       71         ----    ----
;    |      Error         ******
;    |                      75
;  Position
;   the limit
;
;------------------------------------------------------------------------------
;      NON-VOL MEMORY MAP
;------------------------------------------------------------------------------
;
;      00     A0                         D0             Multi-function transmitters
;      01     A0                         D0
;      02     A1                         D0
;      03     A1                         D0
;      04     A2                         D1
;      05     A2                         D1
;      06     A3                         D1
;      07     A3                         D1
;      08     A4                         D2
;      09     A4                         D2
;      0A     A5                         D2
;      0B     A5                         D2
;      0C     A6                         D3
;      0D     A6                         D3
;      0E     A7                         D3
;      0F     A7                         D3
;      10     A9                         D4
;      11     A9                         D4
;      12     A9                         D4
;      13     A9                         D4
;      14     A10                        D5
;      15     A10                        D5
;      16     A11                        D5
;      17     A11                        D5
;      18     B                          D6
;      19     B                          D6
;      1A     C                          D6
;      1B     C                          D6
;      1C     unused             D7
;      1D     unused             D7
;      1E     unused             D7
;      1F     unused             D7
;      20     unused             DTCP          Keyless permanent 4 digit code
;      21     unused             DTCID         Keyless ID code
;      22     unused             DTCR1         Keyless Roll value
;      23     unused             DTCR2
;      24     unused             DTCT          Keyless temporary 4 digit code
;      25     unused             Duration      Keyless temporary duration
;                                                     Upper byte = Mode:  hours/activations
;                                                     Lower byte = # of hours/activations
;      26     unused             Radio type
;                                      77665544 33221100
;                                      00 = CMD     01 = LIGHT
;                                       10 = OPEN/CLOSE/STOP
;       27      unused                Fixed / roll
;                                       Upper word = fixed/roll byte
;                                       Lower word = unused
;       28      CYCLE COUNTER 1ST 16 BITS
;       29      CYCLE COUNTER 2ND 16 BITS
;       2A      VACATION FLAG
;
;               Vacation Flag , Last Operation
;               0000            XXXX  in vacation
;               1111            XXXX out of vacation
;
;       2B      A MEMORY ADDRESS LAST WRITTEN
;       2C      IRLIGHHTADDR 4-22-97
;       2D      Up Limit
;       2E      Pass point counter / Last operating state
;       2F      Down Limit
;
;       30-3F  Force Back trace
;
;-----------------------------------------------------------------------
;       RS232 DATA
;-----------------------------------------------------------------------
;
;       REASON
;       00      COMMAND
;       10      RADIO COMMAND
;       20      FORCE
;       30      AUX OBS
;       40      A REVERSE DELAY
;       50      LIMIT
;       60      EARLY LIMIT
;       70      MOTOR MAX TIME, TIME OUT
;       80      MOTOR COMMANDED OFF RPM CAUSING AREV
;       90      DOWN LIMIT WITH COMMAND HELD
;       A0      DOWN LIMIT WITH THE RADIO HELD
;       B0      RELEASE OF COMMAND OR RADIO AFTER A FORCED
;       UP MOTOR ON DUE TO RPM PULSE WITHG MOTOR OFF
;
;       STATE
;
;       00      AUTOREVERSE DELAY
;       01      TRAVELING UP DIRECTION
;       02      AT THE UP LIMIT AND STOPED
;       03      ERROR RESET
;       04      TRAVELING DOWN DIRECTION
;       05      AT THE DOWN LIMIT
;       06      STOPPED IN MID TRAVEL
;
;
;-----------------------------------------------------------------------
;       DIAG
;-----------------------------------------------------------------------
;
;       1) AOBS SHORTED
;       2) AOBS OPEN / MISS ALIGNED
;       3) COMMAND SHORTED
;       4) PROTECTOR INTERMITTENENT
;       5) CALL DEALER
;          NO RPM IN THE FIRST SECOND
;       6) RPM FORCED A REVERSE
;       7) LIMITS NOT LEARNED YET
;
;
;
;-----------------------------------------------------------------------
;       DOG 2
;-----------------------------------------------------------------------
;
```

```
;       DOG 2 IS A SECONDARY WATCHDOG USED TO
;       RESET THE SYSTEM IF THE LOWEST LEVEL "MAINLOOP"
;       IS NOT REACHED WITHIN A 3 SECOND

;----------------------------------------------------------------------------
;       Conditional Assembly
;----------------------------------------------------------------------------

        GLOBALS ON                          ; Enable a symbol file
Yes                 .equ    1
No                  .equ    0
TwoThirtyThree      .equ    Yes
UseSiminor          .equ    Yes


;
;----------------------------------------------------------------------------
;       EQUATE STATEMENTS
;----------------------------------------------------------------------------

check_sum_value     .equ    065H                ; CRC checksum for ROM code
TIMER_1_EN          .equ    0CH                 ; TMR mask to start timer 1

MOTORTIME           .equ    (27000 / 4)         ; Max. run for motor = 27 sec (4 ms tick)
LAGTIME             .equ    (500 / 4)           ; Delay before learning limits is 0.5 seconds
LEARNTIME           .equ    (50000 / 4)         ; Max. run for motor in learn mode

PWM_CHARGE          .equ    00H                 ; PWM state for old force pots.
LIGHT               .equ    0FFH                ; Flag for light on constantly
LIGHT_ON            .equ    10000000B           ; P0 pin turning on worklight
MOTOR_UP            .equ    01000000B           ; P0 pin turning on the up motor
MOTOR_DN            .equ    00100000B           ; P0 pin turning on the down motor

UP_OUT              .equ    00010000B           ; P3 pin output for up force pot.
DOWN_OUT            .equ    00100000B           ; P3 pin output for down force pot.
DOWN_COMP           .equ    00000001B           ; P0 pin input for down force pot.
UP_COMP             .equ    00000010B           ; P0 pin input for up force pot.

FALSEIR             .equ    00000001B           ; P2 pin for false AOBS output
LINEINPIN           .equ    00010000B           ; P2 pin for reading in AC line

PPointPort          .equ    p2                  ; Port for pass point input
PassPoint           .equ    00001000B           ; Bit mask for pass point input

PhasePrt            .equ    p0                  ; Port for phase control output
PhaseHigh           .equ    00010000B           ; Pin for controlling FET's

CHARGE_SW           .equ    10000000B           ; P3 Pin for charging the wall control
DIS_SW        .equ    01000000B           ; P3 Pin for discharging the wall control
SWITCHES1           .equ    00001000B           ; P0 Pin for first wall control input
SWITCHES2           .equ    00000100B           ; P0 Pin for second wall control input


P01M_INIT           .equ    00000101B           ; set mode p00-p03 in p04-p07 out
P2M_INIT            .equ    01011100B           ; P2M initialization for operation
P2M_POR             .equ    01000000B           ; P2M initialization for output of chip ID
P3M_INIT            .equ    00000011B           ; set port3 p30-p33 input ANALOG mode

P01S_INIT           .equ    10000000B           ; Set init. state as worklight on, motor off
P2S_INIT            .equ    00000110B           ; Init p2 to have LED off
P2S_POR             .equ    00101010B           ; P2 init to output a chip ID (P25, P24, P23, P22)
P3S_INIT            .equ    00000000B           ; Init p3 to have everything off

BLINK_PIN           .equ    00000100B           ; Pin which controls flasher module

P2M_ALLOUTS         .equ    01011100B           ; Pins which need to be refreshed to outputs
P2M_ALLINS          .equ    01011000B           ; Pins which need to be refreshed to inputs

RsPerHalf           .equ    104                 ; RS232 period 1200 Baud half time 416uS
```

```
RsPerFull           .equ   208                  ; RS232 period full time 832us
RsPer1P22           .equ   00                   ; RS232 period 1.22 unit times 1.024ms (00 = 256)

FLASH               .equ   0FFH                 ;
WORKLIGHT           .equ   LIGHT_ON             ; Pin for toggling state of worklight

PPOINTPULSES  .equ   897                  ; Number of RPM pulses between pass points

SetupPos            .equ   (65535 - 20)  ; Setup position -- 2" above pass point

CMD_TEST            .equ   00                   ; States for old wall control routine
WL_TEST             .equ   01
VAC_TEST            .equ   02
CHARGE              .equ   03
RSSTATUS            .equ   04                   ; Hold wall control ckt. in RS232 mode
WALLOFF             .equ   05                   ; Turn off wall control LED for blinks

AUTO_REV            .equ   00H                  ; States for GDO state machine
UP_DIRECTION  .equ   01H
UP_POSITION   .equ   02H
DN_DIRECTION  .equ   04H
DN_POSITION   .equ   05H
STOP                .equ   06H
CMD_SW              .equ   01H                  ; Flags for switches hit
LIGHT_SW            .equ   02H
VAC_SW              .equ   04H

TRUE                .equ   0FFH                 ; Generic constants
FALSE               .equ   00H

FIXED_MODE          .equ   10101010b                   ;Fixed mode radio
ROLL_MODE           .equ   01010101b                   ;Rolling mode radio
FIXED_TEST          .equ   00000000b                   ;Unsure of mode -- test fixed
ROLL_TEST           .equ   00000001b                   ;Unsure of mode -- test roll
FIXED_MASK          .equ   FIXED_TEST                  ;Bit mask for fixed mode
ROLL_MASK           .equ   ROLL_TEST                   ;Bit mask for rolling mode

FIXTHR        .equ   03H                  ;Fixed code decision threshold
DTHR                .equ   02H                         ;Rolling code decision threshold
FIXSYNC             .equ   08H                         ;Fixed code sync threshold
DSYNC               .equ   04H                         ;Rolling code sync threshold
FIXBITS             .equ   11                          ;Fixed code number of bits
DBITS               .equ   21                          ;Rolling code number of bits

EQUAL               .equ   00                          ;Counter compare result constants
BACKWIN             .equ   7FH                         ;
FWDWIN        .equ   80H                  ;
OUTOFWIN            .equ   0FFH                        ;

AddressCounter      .equ   27H
AddressAPointer     .equ   2BH

CYCCOUNT            .equ   28H

TOUCHID             .equ   21H                         ;Touch code ID
TOUCHROLL           .equ   22H                         ;Touch code roll value
TOUCHPERM           .equ   20H                         ;Touch code permanent password
TOUCHTEMP           .equ   24H                         ;Touch code temporary password
DURAT               .equ   25H                         ;Touch code temp. duration

VERSIONNUM          .equ   088H                        ;Version:  PRO7000 V2.8
;4-22-97
IRLIGHTADDR         .EQU   2CH                         ;work light feature on or off
DISABLED            .EQU   00H                         ;00 = disabled, FF = enabled
;
RTYPEADDR           .equ   26H                         ;Radio transmitter type
VACATIONADDR  .equ   2AH
MODEADDR            .equ   27H                         ;Rolling/Fixed mode in EEPROM
                                                       ;High byte = don't care (now)
                                                       ;Low byte = RadioMode flag
UPLIMADDR             .equ   2DH                       ;Address of up limit
LASTSTATEADDR . qu   2EH                      ;Address of last state
DNLIMADDR             .equ   2FH                       ;Address of down limit

NOEECOMM              .equ   01111111b                 ;Flag:  skip radio read/write
NOINT                 .equ   10000000b                 ;Flag:  skip radio int rrupts

RDROPTIME             .equ   125                       ;Radio drop-out time:  0.5s

LRNOCS         .equ   0AAH                    ;Learn open/close/stop
BRECEIVED             .equ   077H                      ;B code received flag
LRNLIGHT              .equ   0BBH                      ;Light command trans.
LRNTEMP               .equ   0CCH                      ;Learn touchcode temporary
LRNDURTN              .equ   0DDH                      ;Learn t.c. temp. duration
REGLEARN              .equ   0EEH                      ;Regular learn mode
NORMAL         .equ   00H                     ;Normal command trans.

ENTER                 .equ   00H                       ;Touch code ENTER key
POUND                 .equ   01H                       ;Touch code # key
STAR                  .equ   02H                       ;Touch code * key

ACTIVATIONS           .equ   0AAH                      ;Number of activations mode
HOURS                 .equ   055H                      ;Number of hours mode

      ;Flags for Ramp Flag Register

STILL                 .equ   00H                       ; Motor not moving
RAMPUP         .equ   0AAH                    ; Ramp speed up to maximum
RAMPDOWN              .equ   0FFH                      ; Slow down the motor to minimum
FULLSPEED             .equ   0CCH                      ; Running at full speed

UPSLOWSTART           .equ   200                       ; Distance (in pulses) from limit when slow-
down
DNSLOWSTART           .equ   220                       ; of GDO motor starts (for up and down
direction)

BACKOFF               .equ   16                        ; Distance (in pulses) to back trolley off of
floor
                                                       ; when learning limits by reversing off of
floor

SHORTDOOR             .equ    936                      ; Travel distance (in pulses) that
discriminates a
                                                       ; one piece door (slow travel) from a normal
door
                                                       ; (normal travel)  (Roughly 78")

;-----------------------------------------------------------------------
;      PERIODS
;-----------------------------------------------------------------------


AUTO_REV_TIME         .equ   124                       ; (4 ms prescale)
MIN_COUNT             .equ   02H                       ; pwm start point
TOTAL_PWM_COUNT       .equ   03FH                      ; pwm end = start + 2*total-1
FLASH_TIME            .equ   61                        ; 0.25 sec flash time

      ;4.5 MINUTE USA LIGHT TIMER

USA_LIGHT_HI  .equ   080H                     ; 4.5 MIN
USA_LIGHT_LO  .equ   0BEH                     ; 4.5 MIN

      ;2.5 MINUTE EUROPEAN LIGHT TIMER

EURO_LIGHT_HI         .equ   047H                      ; 2.5 MIN
EURO_LIGHT_LO         .equ   086H                      ; 2.5 MIN

ONE_SEC               .equ   0F4H                      ; WITH A /4 IN FRONT
```

```
CMD_MAKE            .equ   8                       ; cycle count *10mS
CMD_BREAK           .equ   (255-8)
LIGHT_MAKE          .equ   8                       ; cycle count *11mS
LIGHT_BREAK  .equ   (255-8)
VAC_MAKE_OUT .equ   4                       ; cycle count *100mS
VAC_BREAK_OUT       .equ   (255-4)
VAC_MAKE_IN  .equ   2
VAC_BREAK_IN .equ   (255-2)

VAC_DEL             .equ   8                       ; Delay 16 ms for vacation
CMD_DEL_EX          .equ   6                       ; Delay 12 ms ( 5*2 + 2)
VAC_DEL_EX          .equ   50                      ; Delay 100 ms


;*****************************************************************
;      PREDEFINED REG
;*****************************************************************
ALL_ON_IMR          .equ   00111101b               ; turn on int for timers rpm auxobs radio
RETURN_IMR          .equ   00111100b               ; return on the IMR

RadioImr            .equ   00000001b               ; turn on the radio only

;------------------------------------------------------------------------
;      GLOBAL REGISTERS
;------------------------------------------------------------------------

STATUS               .equ  04H                     ; CMD_TEST 00
                                                   ; WL_TEST 01
                                                   ; VAC_TEST 02
                                                   ; CHARGE 03

STATE               .equ   05H                     ; state register
LineCtr             .equ   06H                     ;
RampFlag            .equ   07H                     ; Ramp up, ramp down, or stop
AUTO_DELAY          .equ   08H
LinePer             .equ   09H                     ; Period of AC line coming in
MOTOR_TIMER_HI      .equ   0AH
MOTOR_TIMER_LO      .equ   0BH
MOTOR_TIMER  .equ   0AH
LIGHT_TIMER_HI      .equ   0CH
LIGHT_TIMER_LO      .equ   0DH
LIGHT_TIMER  .equ   0CH
AOBSF               .equ   0EH
PrevPass            .equ   0FH



CHECK_GRP           .equ   10H
check_sum           .equ   r0                      ; check sum pointer
rom_data            .equ   r1
test_adr_hi  .equ   r2
test_adr_lo  .equ   r3
test_adr            .equ   rr2
CHECK_SUM           .equ   CHECK_GRP+0             ; check sum reg for por
ROM_DATA            .equ   CHECK_GRP+1      ; data read
LIM_TEST_HI         .equ   CHECK_GRP+0             ; Compare registers for measuring
LIM_TEST_LO         .equ   CHECK_GRP+1             ; distance to limit
LIM_TEST            .equ   CHECK_GRP+0             ;
AUXLEARNSW   .equ   CHECK_GRP+2       ;
RRTO                .equ   CHECK_GRP+3      ;
RPM_ACOUNT   .equ   CHECK_GRP+4       ; to test for active rpm
RS_COUNTER          .equ   CHECK_GRP+5      ; rs232 byte counter
RS232DAT            .equ   CHECK_GRP+6      ; rs232 data

RADIO_CMD           .equ   CHECK_GRP+7      ; radio command
R_DEAD_TIME  .equ   CHECK_GRP+8       ;
FAULT               .equ   CHECK_GRP+9      ;
VACFLAG             .equ   CHECK_GRP+10            ; VACATION mode flag
VACFLASH            .equ   CHECK_GRP+11
```

```
VACCHANGE           .equ   CHECK_GRP+12
FAULTTIME           .equ   CHECK_GRP+13
FORCE_IGNORE  . qu  CHECK_GRP+14
FAULTCODE           .equ   CHECK_GRP+15



TIMER_GROUP   .equ  20H
position_hi         .equ   r0
position_lo         .equ   r1
position            .equ   rr0
up_limit_hi         .equ   r2
up_limit_lo         .equ   r3
up_limit            .equ   rr2
switch_delay  .equ  r4
obs_count           .equ   r6
rscommand           .equ   r9
rs_temp_hi          .equ   r10
rs_temp_lo          .equ   r11
rs_temp             .equ   rr10

POSITION_HI         .equ   TIMER_GROUP+0
POSITION_LO         .equ   TIMER_GROUP+1
POSITION            .equ   TIMER_GROUP+0
UP_LIMIT_HI         .equ   TIMER_GROUP+2
UP_LIMIT_LO         .equ   TIMER_GROUP+3
UP_LIMIT            .equ   TIMER_GROUP+2
SWITCH_DELAY  .equ  TIMER_GROUP+4
OnePass             .equ   TIMER_GROUP+5
OBS_COUNT           .equ   TIMER_GROUP+6
RsMode              .equ   TIMER_GROUP+7
Divisor             .equ   TIMER_GROUP+8        ; Number to divide by
RSCOMMAND           .equ   TIMER_GROUP+9
RS_TEMP_HI          .equ   TIMER_GROUP+10
RS_TEMP_LO          .equ   TIMER_GROUP+11
RS_TEMP             .equ   TIMER_GROUP+10
PowerLevel          .equ   TIMER_GROUP+12               ; Current step in 20-step phase ramp-up
PhaseTMR            .equ   TIMER_GROUP+13               ; Timer for turning on and off phase control
PhaseTime           .equ   TIMER_GROUP+14               ; Current time reload value for phase timer
MaxSpeed            .equ   TIMER_GROUP+15               ; Maximum speed for this kind of door

;*************************************************************
; LEARN EE GROUP FOR LOOPS ECT
;*************************************************************
LEARNEE_GRP   .equ  30H                 ;
TEMPH         .equ  LEARNEE_GRP    ;
TEMPL         .equ  LEARNEE_GRP+1       ;
P2M_SHADOW          .equ   LEARNEE_GRP+2        ; Readable shadow of P2M register
LEARNDB             .equ   LEARNEE_GRP+3        ; learn debouncer
LEARNT              .equ   LEARNEE_GRP-4        ; learn timer
ERASET              .equ   LEARNEE_GRP+5        ; erase timer
MTEMPH              .equ   LEARNEE_GRP+6    ; memory temp
MTEMPL              .equ   LEARNEE_GRP+7        ; memory temp
MTEMP         .equ  LEARNEE_GRP+8       ; memory temp
SERIAL              .equ   LEARNEE_GRP+9        ; data to & from nonvol memory
ADDRESS             .equ   LEARNEE_GRP+10       ; address for the serial nonvol memory
ZZWIN         .equ  LEARNEE_GRP+11      ; radio 00 code window
T0_OFLOW            .equ   LEARNEE_GRP+12       ; Third byte of T0 counter
T0EXT         .equ  LEARNEE_GRP+13      ; t0 extend dec'd every T0 int
T0EXTWORD           .equ   LEARNEE_GRP+12       ; Word-wide T0 extension
T125MS              .equ   LEARNEE_GRP+14       ; 125mS counter
SKIPRADIO           .equ   LEARNEE_GRP+15       ; flag to skip radio read, write if
                                                ; learn or vacation talking to it

temph         .equ  r0                  ;
templ         .equ  r1                  ;
learndb             .equ   r3                   ; learn debouncer
learnt              .equ   r4                   ; learn timer
eraset              .equ   r5                   ; erase timer
mtemph              .equ   r6                   ; memory temp
```

```
mtempl             .equ  r7                                  ; memory temp
mtemp        . qu  r8                              ; memory temp
serial       .equ  r9                              ; data to and from nonvol mem
address            .equ  r10                                 ; addr for serial nonvol memory
zzwin        .equ  r11                             ;
t0_oflow           .equ  r12                                 ; Overflow counter for T0
t0ext        .equ  r13                             ; t0 extend dec'd every T0 int
t0extword          .equ  rr12                                ; Word-wide T0 extension
t125ms             .equ  r14                                 ; 125mS counter
skipradio          .equ  r15                                 ; flag to skip radio read, write if
                                                             ; learn or vacation talking to it



FORCE_GROUP        .equ  40H
dnforce            .equ  r0
upforce            .equ  r1
loopreg            .equ  r3
up_force_hi  .equ  r4
up_force_lo  .equ  r5
up_force           .equ  rr4
dn_force_hi  .equ  r6
dn_force_lo  .equ  r7
dn_force           .equ  rr6
force_add_hi .equ  r8
force_add_lo .equ  r9
force_add          .equ  rr8
up_temp            .equ  r10
dn_temp            .equ  r11
pot_count          .equ  r12
force_temp_of.equ  r13
force_temp_hi.equ  r14
force_temp_lo.equ  r15

DNFORCE            .equ  40H
UPFORCE            .equ  41H
AOBSTEST           .equ  42H
LoopReg            .equ  43H
UP_FORCE_HI  .equ  44H
UP_FORCE_LO  .equ  45H
DN_FORCE_HI  .equ  46H
DN_FORCE_LO  .equ  47H
UP_TEMP            .equ  4AH
DN_TEMP            .equ  4BH
POT_COUNT          .equ  4CH
FORCE_TEMP_OF.equ  4CH
FORCE_TEMP_HI      .equ  4EH
FORCE_TEMP_LO      .equ  4FH



RPM_GROUP          .equ  50H

rtypes2            .equ  r0
stackflag          .equ  r1
rpm_temp_of        .equ  r2
rpm_temp_hi  .equ  r3
rpm_t mp_hiword    .equ  rr2
rpm_temp_lo  .equ  r4
rpm_past_hi  .equ  r5
rpm_past_lo  .equ  r6
rpm_period_hi      .equ  r7
rpm_period_lo      .equ  r8
divcounter         .equ  r11                                 ; Counter for dividing RPM time
rpm_count          .equ  r12
rpm_time_out .equ  r13

RTypes2            .equ  RPM_GROUP+0
STACKFLAG          .equ  RPM_GROUP+1
```

```
RPM_TEMP_OF        .equ  RPM_GROUP+2             ; Overflow for RPM Time
RPM_TEMP_HI  .equ  RPM_GROUP+3             ;
RPM_TEMP_HWORD     .equ  RPM_GROUP+2             ; High word of RPM Time
RPM_TEMP_LO  .equ  RPM_GROUP+4
RPM_PAST_HI  . qu  RPM_GROUP+5
RPM_PAST_LO  .equ  RPM_GROUP+6
RPM_PERIOD_HI      .equ  RPM_GROUP+7
RPM_PERIOD_LO      .equ  RPM_GROUP+8
DN_LIMIT_HI        .equ  RPM_GROUP+9             ;
DN_LIMIT_LO        .equ  RPM_GROUP+10      ;
DIVCOUNTER         .equ  RPM_GROUP+11      ; Counter for dividing RPM time
RPM_FILTER         .equ  RPM_GROUP+11      ; DOUBLE MAPPED register for filtering signal
RPM_COUNT          .equ  RPM_GROUP+12
RPM_TIME_OUT .equ  RPM_GROUP+13
BLINK_HI           .equ  RPM_GROUP+14      ; Blink timer for flashing the
BLINK_LO           .equ  RPM_GROUP+15      ; about-to-travel warning light
BLINK              .equ  RPM_GROUP+14      ; Word-wise blink timer


;*****************************************************************
; RADIO GROUP
;*****************************************************************
RadioGroup   .equ  60H                     ;
RTemp              .equ  RadioGroup              ; radio temp storage
RTempH       .equ  RadioGroup+1      ; radio temp storage high
RTempL       .equ  RadioGroup+2      ; radio temp storage low
RTimeAH            .equ  RadioGroup+3      ; radio active time high byte
RTimeAL            .equ  RadioGroup+4      ; radio active time low byte
RTimeIH            .equ  RadioGroup+5      ; radio inactive time high byte
RTimeIL            .equ  RadioGroup+6      ; radio inactive time low byte
Radio1H            .equ  RadioGroup+7      ; sync 1 code storage
Radio1L            .equ  RadioGroup+8      ; sync 1 code storage
RadioC       .equ  RadioGroup+9      ; radio word count
PointerH           .equ  RadioGroup+10     ;
PointerL           .equ  RadioGroup+11     ;
AddValueH          .equ  RadioGroup+12     ;
AddValueL          .equ  RadioGroup+13     ;
Radio3H            .equ  RadioGroup+14     ; sync 3 code storage
Radio3L            .equ  RadioGroup+15     ; sync 3 code storage
rtemp              .equ  r0                      ; radio temp storage
rtemph       .equ  r1                      ; radio temp storage high
rtempl       .equ  r2                      ; radio temp storage low
rtimeah            .equ  r3                      ; radio active time high byte
rtimeal            .equ  r4                      ; radio active time low byte
rtimeih            .equ  r5                      ; radio inactive time high byte
rtimeil            .equ  r6                      ; radio inactive time low byte
radio1h            .equ  r7                      ; sync 1 code storage
radio1l            .equ  r8                      ; sync 1 code storage
radioc       .equ  r9                      ; radio word count
pointerh           .equ  r10                     ;
pointerl           .equ  r11                     ;
pointer            .equ  rr10                    ; Overall pointer for ROM
addvalueh          .equ  r12                     ;
addvaluel          .equ  r13                     ;
radio3h            .equ  r14                     ; sync 3 code storage
radio3l            .equ  r15                     ; sync 3 code storage
w2                 .equ  rr14                    ; For Siminor revision


CounterGroup .equ  070h                    ; counter group
TestReg            .equ  CounterGroup      ; Test area when dividing
BitMask            .equ  CounterGroup+01         ; Mask for transmitters
LastMatch          .equ  CounterGroup+02         ; last matching code address
LoopCount          .equ  CounterGroup+03         ; loop counter
CounterA           .equ  CounterGroup+04         ; counter translation MSB
CounterB           .equ  CounterGroup+05         ;
CounterC           .equ  CounterGroup+06         ;
```

```
CounterD            .equ   CounterGroup+07          ; counter translation LSB
MirrorA             .equ   CounterGroup+08          ; back translation MSB
MirrorB             .equ   CounterGroup+09          ;
MirrorC             .equ   CounterGroup+010         ;
MirrorD             .equ   CounterGroup+011         ; back translation LSB
COUNT1H             .equ   CounterGroup+012         ; rec iv d count
COUNT1L             .equ   CounterGroup+013
COUNT3H             .equ   CounterGroup+014
COUNT3L             .equ   CounterGroup+015

loopcount           .equ   r3                       ;
countera            .equ   r4                       ;
counterb            .equ   r5                       ;
counterc            .equ   r6                       ;
counterd            .equ   r7                       ;
mirrora             .equ   r8                       ;
mirrorb             .equ   r9                       ;
mirrorc             .equ   r10                      ;
mirrord             .equ   r11                      ;

Radio2Group         .equ   080H

PREVFIX             .equ   Radio2Group + 0
PREVTMP             .equ   Radio2Group + 1
ROLLBIT             .equ   Radio2Group + 2
RTimeDH             .equ   Radio2Group + 3
RTimeDL             .equ   Radio2Group + 4
RTimePH             .equ   Radio2Group + 5
RTimePL             .equ   Radio2Group + 6
ID_B                .equ   Radio2Group + 7
SW_B                .equ   Radio2Group + 8
RADIOBIT            .equ   Radio2Group + 9
RadioTimeOut  .equ   Radio2Group + 10
RadioMode           .equ   Radio2Group + 11         ;Fixed or rolling mode
BitThresh           .equ   Radio2Group + 12         ;Bit decision threshold
SyncThresh          .equ   Radio2Group + 13         ;Sync pulse decision threshold
MaxBits             .equ   Radio2Group + 14         ;Maximum number of bits
RFlag               .equ   Radio2Group + 15         ;Radio flags

prevfix             .equ   r0
prevtmp             .equ   r1
rollbit             .equ   r2
id_b                .equ   r7
sw_b                .equ   r8
radiobit            .equ   r9
radiotimeout  .equ   r10
radiomode           .equ   r11
rflag               .equ   r15


OrginalGroup  .equ   90H
SW_DATA             .equ   OrginalGroup+0
ONEP2               .equ   OrginalGroup+1           ; 1.2 SEC TIMER TICK .125
LAST_CMD            .equ   OrginalGroup+2           ; LAST COMMAND FROM
                                                    ; = 55 WALL CONTROL
                                                    ; = 00 RADIO
CodeFlag            .equ   OrginalGroup+3           ; Radio code type flag
                                                    ; FF = Learning open/close/stop
                                                    ; 77 = b code
                                                    ; AA = open/close/stop code
                                                    ; 55 = Light control transmitter
                                                    ; 00 = Command or unknown
RPMONES             .equ   OrginalGroup+4           ; RPM Pulse One Sec. Disable
RPMCLEAR            .equ   OrginalGroup+5           ; RPM PULSE CLEAR & TEST TIMER
FAREVFLAG           .equ   OrginalGroup+6           ; RPM FORCED AREV FLAG
                                                    ; 88H FOR A FORCED REVERSE

FLASH_FLAG          .equ   OrginalGroup+7
FLASH_DELAY   .equ   OrginalGroup+8
```

```
REASON        .equ   OrginalGroup+9
FLASH_COUNTER       .equ   OrginalGroup+10
RadioTypes          .equ   OrginalGroup+11          ; Types for one page of tx's
LIGHT_FLAG          .equ   OrginalGroup+12
CMD_DEB             .equ   OrginalGroup+13
LIGHT_DEB           .equ   OrginalGroup+14
VAC_DEB             .equ   OrginalGroup+15

NextGroup           .equ   0A0H
SDISABLE            .equ   NextGroup+0        ; system disable timer
PRADIO3H            .equ   NextGroup+1        ; 3 mS code storage high byte
PRADIO3L            .equ   NextGroup+2        ; 3 mS code storage low byte
PRADIO1H            .equ   NextGroup+3        ; 1 mS code storage high byte
PRADIO1L            .equ   NextGroup+4        ; 1 mS code storage low byte
RTO                 .equ   NextGroup+5        ; radio time out
;RFlag              .equ   NextGroup+6        ; radio flags
EnableWorkLight     .equ   NextGroup+6               ;4-22-97 work light function on or off?
RINFILTER           .equ   NextGroup+7        ; radio input filter

LIGHT1S             .equ   NextGroup+8        ; light timer for 1second flash
DOG2                .equ   NextGroup+9        ; second watchdog
FAULTFLAG           .equ   NextGroup+10       ; flag for fault blink, no rad. blink
MOTDEL              .equ   NextGroup+11       ; motor time delay
PPOINT_DEB          .equ   NextGroup+12       ; Pass Point debouncer
DELAYC              .equ   NextGroup+13       ; for the time delay for command
L_A_C               .equ   NextGroup+14       ; Limits are changing register
CMP                 .equ   NextGroup+15       ; Counter compare result

BACKUP_GRP          .equ   0B0H
PCounterA           .equ   BACKUP_GRP
PCounterB           .equ   BACKUP_GRP+1
PCounterC           .equ   BACKUP_GRP+2
PCounterD           .equ   BACKUP_GRP+3
HOUR_TIMER          .equ   BACKUP_GRP+4
HOUR_TIMER_HI .equ   BACKUP_GRP+4
HOUR_TIMER_LO .equ   BACKUP_GRP+5
PassCounter         .equ   BACKUP_GRP+6
STACKREASON         .equ   BACKUP_GRP+7
FirstRun            .equ   BACKUP_GRP+8       ; Flag for first operation after power-up
MinSpeed            .equ   BACKUP_GRP+9
BRPM_COUNT          .equ   BACKUP_GRP+10
BRPM_TIME_OUT       .equ   BACKUP_GRP+11
BFORCE_IGNORE       .equ   BACKUP_GRP+12
BAUTO_DELAY   .equ   BACKUP_GRP+13
BCMD_DEB            .equ   BACKUP_GRP+14
BSTATE              .equ   BACKUP_GRP+15

;       Double-mapped registers for M6800 test
COUNT_HI            .equ   BRPM_COUNT
COUNT_LO            .equ   BRPM_TIME_OUT
COUNT               .equ   BFORCE_IGNORE
REGTEMP             .equ   BAUTO_DELAY
REGTEMP2            .equ   BCMD_DEB

;       Double-mapped registers for Siminor Code Reception

CodeT0        .equ   COUNT1L                  ; Binary radio code received
CodeT1        .equ   Radio1L
CodeT2        .equ   MirrorC
CodeT3        .equ   MirrorD
CodeT4        .equ   COUNT3H
CodeT5        .equ   COUNT3L

Ix                  .equ   COUNT1H                   ; Index per Siminor's code

W1High        .equ   AddValueH                ; Word 1 per Siminor's code
W1Low               .equ   AddValueL                 ; description
w1high        .equ   addvalueh
w1low               .equ   addvaluel
```

```
W2High        . qu   Radio3H                ; Word 2 per Siminor's code
W2Low                .equ   Radio3L               ; description
w2high        .equ   radio3h
w2low                .equ   radio3l

STACKTOP             .equ   238                   ; start of the stack
STACKEND             .equ   0C0H                  ; end of the stack

RS232IP              .equ   P0                    ; RS232 input port
RS232IM              .equ   SWITCHES1             ; RS232 mask

csh                  .equ   10000000B             ; chip select high for the 93c46
csl                  .equ   ~csh                  ; chip select low for 93c46
clockh               .equ   01000000B             ; clock high for 93c46
clockl               .equ   ~clockh               ; clock low for 93c46
doh                  .equ   00100000B             ; data out high for 93c46
dol                  .equ   ~doh                  ; data out low for 93c46
ledh                 .equ   00000010B             ; turn the led pin high "off
ledl                 .equ   ~ledh           ; turn the led pin low "on
psmask               .equ   01000000B             ; mask for the program switch
csport               .equ   P2                    ; chip select port
dioport              .equ   P2                    ; data i/o port
clkport              .equ   P2                    ; clock port
ledport              .equ   P2                    ; led port
psport               .equ   P2                    ; program switch port

WATCHDOG_GROUP       .equ   0FH
pcon                 .equ   r0
smr                  .equ   r11
wdtmr                .equ   r15

;       .IF    TwoThirtyThree
;
;WDT    .macro
;       .byte  5fh
;       .endm
;
;       .ELSE
;
;WDT    .macro
;       xor    P1, #00000001b                     ; Kick external watchdog
;       .endm
;
;       .ENDIF

FILL    .macro
        .byte  0FFh
        .endm

FILL10 .macro
        FILL
        FILL
        FILL
        FILL
        FILL
        FILL
        FILL
        FILL
        FILL
        FILL
        .endm

FILL100        .macro
        FILL10
        FILL10
        FILL10
        FILL10
```

```
      FILL10
      FILL10
      FILL10
      FILL10
      FILL10
      FILL10
      .endm

FILL1000      .macro
      FILL100
      FILL100
      FILL100
      FILL100
      FILL100
      FILL100
      FILL100
      FILL100
      FILL100
      FILL100
      .endm

TRAP   .macro
      jp     start
      jp     start
      jp     start
      jp     start
      jp     start
      .endm
TRAP10 .macro
      TRAP
      TRAP
      TRAP
      TRAP
      TRAP
      TRAP
      TRAP
      TRAP
      TRAP
      TRAP
      .endm

SetRpToRadio2Group   .macro
       .byte  031H
       .byte  080H
                          .endm


;*****************************************************************
;*
;* Interrupt Vector Table
;*
;*****************************************************************

       .org    0000H

       .IF     TwoThirtyThree

       .word  RADIO_INT                        ;IRQ0
       .word  000CH                            ;IRQ1, P3.3
       .word  RPM                              ;IRQ2, P3.1
       .word  AUX_OBS                          ;IRQ3, P3.0
       .word  TIMERUD                          ;IRQ4, T0
       .word  RS232                            ;IRQ5, T1

       .ELSE

       .word  RADIO_INT                        ;IRQ0
       .word  RADIO_INT                        ;IRQ1, P3.3
       .word  RPM                              ;IRQ2, P3.1
      .word  AUX_OBS                           ;IRQ3, P3.0
      .word  TIMERUD                           ;IRQ4, T0
      .word  000CH                             ;IRQ5, T1

      .ENDIF

      .page
      .org    000CH
      jp     START                             ;jmps to start at location 0101, 0202 etc

;-----------------------------------------------------------------------------
;     RS232 DATA ROUTINES
;
;     RS_COUNTER REGISTER:
;     0000XXXX - 0011XXXX Input byte counter (inputting bytes 1-4)
;     00XX0000               Waiting for a start bit
;     00XX0001 - XXXX1001 Input bit counter (Bits 1-9, including stop)
;     00XX1111               Idle -- whole byte received
;
;     1000XXXX - 1111XXXX Output byte counter (outputting bytes 1-8)
;     1XXX0000               Tell the routine to output a byte
;     1XXX0001 - 1XXX1001 Outputting a byte (Bits 1-9, including stop)
;     1XXX1111               Idle -- whole byte output
;
;-----------------------------------------------------------------------------

OutputMode:

      tm     RS_COUNTER, #00001111B                    ; Check for outputting start bit
      jr     z, OutputStart                            ;

      tcm    RS_COUNTER, #00001001B                    ; Check for outputting stop bit
      jr     z, OutputStop                     ; (bit 9), if so, don't increment

OutputData:

      scf                                              ; Set carry to ensure high stop bit
      rrc    RS232DAT                                  ; Test the bit for output
      jr     c, OutputHigh                     ;

OutputLow:

      and    p3, #~CHARGE_SW                           ; Turn off the pull-up
      or     P3, #DIS_SW                               ; Turn on the pull-down
      jr     DataBitDone                               ;

OutputStart:

      ld     T1,#RsPerFull                     ; Set the timer to a full bit period
      ld     TMR, #00001110B                           ; Load the full time period
      and    p3, #~CHARGE_SW                           ; Send a start bit
      or     P3, #DIS_SW                               ;
      inc    RS_COUNTER                                ; Set the counter to first bit
      iret                                             ;

OutputHigh:

      and    p3, #~DIS_SW                      ; Turn off the pull-down
      or     P3, #CHARGE_SW                            ; Turn on the pull-up

DataBitDone:

      inc    RS_COUNTER                                ; Advance to the next data bit
      iret                                             ;

OutputStop:

      and    p3, #~DIS_SW                      ; Output a stop (high) bit
      or     P3, #CHARGE_SW                            ;
      or      RS_COUNTER, #00001111B              ; Set the flag for word being done
      cp      RS_COUNTER, #11111111B              ; T st for last output byte
      jr      nz, MoreOutput                      ; If not, wait for more output
      clr     RS_COUNTER                          ; Start waiting for input bytes
MoreOutput:
RSExit:
      iret                                        ;

RS232:

      cp      RsMode, #00                         ; Check for in RS232 mode,
      jr      nz, InRsMode                   ; If so, keep receiving data
      cp      STATUS, #CHARGE                     ; Else, only receive data when
      jr      nz, WallModeBad                     ; charging the wall contol

InRsMode:

      tcm     RS_COUNTER, #00001111B              ; Test for idle state
      jr      z, RSExit                           ; If so, don't do anything

      tm      RS_COUNTER, #11000000B              ; test for input or output mode
      jr      nz, OutputMode

RSInput:

      tm      RS_COUNTER, #00001111B              ; Check for waiting for start
      jr      z, WaitForStart                     ; If so, test for start bit

      tcm     RS_COUNTER, #00001001B              ; Test for receiving the stop bit
      jr      z, StopBit                          ; If so, end the word

      scf                                       ' ; Initially set the data in bit
      tm      RS232IP,#RS232IM                    ; Check for high or low bit at input
      jr      nz, GotRsBit                   ; If high, leave carry high

      rcf                                         ; Input bit was low

GotRsBit:

      rrc     RS232DAT                            ; Shift the bit into the byte
      inc     RS_COUNTER                          ; Advance to the next bit
      iret

StopBit:

      tm      RS232IP,#RS232IM                    ; Test for a valid stop bit
      jr      z, DataBad                          ; If invalid, throw out the word

DataGood:

      tm      RS_COUNTER, #11110000B              ; If we're not reading the first word,
      jr      nz, IsData                          ; then this is not a command
      ld      RSCOMMAND, RS232DAT            ; Load the new command word
IsData:
      or      RS_COUNTER, #00001111B              ; Indicate idle at end of word
      iret

WallModeBad:

      clr     RS_COUNTER                          ; Reset the RS232 state

DataBad:

      and     RS_COUNTER, #00110000B              ; Clear the byte counter
      iret

WaitForStart:

      tm      RS232IP,#RS232IM                    ; Check for a start bit
```

```
        jr      nz, NoStartBit                          ; If high, keep waiting

        inc     RS_COUNTER                              ; Set to receive bit 1
        ld      T1, #RsPer1P22                          ; Long time until next sample
        ld      TMR, #00001110B                         ; Load the timer
        ld      T1, #RsPerFull                          ; Sample at 1X afterwards
        iret                                            ;

NoStartBit:

        ld      T1, #RsPerHalf                          ; Sample at 2X for start bit
        iret

;-----------------------------------------------------------------------------
;       Set the worklight timer to 4.5 minutes for 60Hz line
;       and 2.5 minutes for 50 Hz line
;-----------------------------------------------------------------------------
SetVarLight:
        cp      LinePer, #36                  ; Test for 50Hz or 60Hz
        jr      uge, EuroLight                          ; Load the proper table
USALight:
        ld      LIGHT_TIMER_HI,#USA_LIGHT_HI            ; set the light period
        ld      LIGHT_TIMER_LO,#USA_LIGHT_LO            ;
        ret                                             ; Return
EuroLight:
        ld      LIGHT_TIMER_HI,#EURO_LIGHT_HI           ; set the light period
        ld      LIGHT_TIMER_LO,#EURO_LIGHT_LO           ;
        ret                                             ; Return
;-----------------------------------------------------------------------------
;       THIS THE AUXILARY OBSTRUCTION INTERRUPT ROUTINE
;-----------------------------------------------------------------------------
AUX_OBS:
        ld      OBS_COUNT,#11                           ; reset pulse counter (no obstruction)
        and     imr,#11110111b                          ; turn off the interupt for up to 500uS
        ld      AOBSTEST,#11                  ; reset the test timer
        or      AOBSF,#00000010B                        ; set the flag for got a aobs
        and     AOBSF,#11011111B                        ; Clear the bad aobs flag
        iret                                            ; return from int

;------------------------------------------------
;       Test for the presence of a blinker module
;------------------------------------------------
LookForFlasher:
        and     P2M_SHADOW, #~BLINK_PIN                 ;Set high for autolatch test
        ld      P2M, P2M_SHADOW                         ;
        or      P2, #BLINK_PIN                          ;
        or      P2M_SHADOW, #BLINK_PIN                  ;Look for Flasher module
        ld      P2M, P2M_SHADOW                         ;
        ret

        ; Fill 41 bytes of unused memory

        FILL10
        FILL10
        FILL10
        FILL10
        FILL

;*********************************************************************
; REGISTER INITILIZATION
;*********************************************************************

        .org    0101H                                   ; address has both bytes the same
start:
START: di                                     ; turn off the interrupt for init

        .IF     TwoThirtyThree
```

```
	ld	RP,#WATCHDOG_GROUP
	ld	wdtmr,#00001111B			; rc dog 100mS

	.ELSE

	clr	P1

	.ENDIF

	WDT					; kick the dog
	clr	RP				; clear the register pointer

;*********************************************************************
; PORT INITILIZATION
;*********************************************************************


	ld	P0,#P01S_INIT		; RESET all ports
	ld	P2,#P2S_POR		; Output the chip ID code
	ld	P3,#P3S_INIT		;
	ld	P01M,#P01M_INIT			; set mode p00-p03 out p04-p07in
	ld	P3M,#P3M_INIT			; set port3 p30-p33 input analog mode
						; p34-p37 outputs
	ld	P2M,#P2M_POR		; set port 2 mode for chip ID out
;*********************************************************************
;* Internal RAM Test and Reset All RAM = mS *
;*********************************************************************
	srp	#0F0h				; point to control group use stack
	ld	r15,#4				;r15= pointer (minimum of RAM)
write_again:
	WDT					; KICK THE DOG
	ld	r14,#1
write_again1:
	ld	@r15,r14			;write 1,2,4,8,10,20,40,80
	cp	r14,@r15			;then compare
	jr	ne,system_error
	rl	r14
	jr	nc,write_again1
	clr	@r15				;write RAM(r5)=0 to memory
	inc	r15
	cp	r15,#240
	jr	ult,write_again



;*********************************************************************
;*	Checksum Test			*
;*********************************************************************
CHECKSUMTEST:
	srp	#CHECK_GRP
	ld	test_adr_hi,#01FH
	ld	test_adr_lo,#0FFH		;maximum address=fffh
add_sum:
	WDT					; KICK THE DOG
	ldc	rom_data,@test_adr		;read ROM code one by one
	add	check_sum,rom_data		;add it to checksum register
	decw	test_adr			;increment ROM address
	jr	nz,add_sum		;address=0 ?
	cp	check_sum,#check_sum_value
	jr	z,system_ok		;check final checksum = 00 ?
system_error:
	and	ledport,#led1		; turn on the LED to indicate fault
	jr	system_error

	.byte	256-check_sum_value
system_ok:
        WDT                                     ; kick th  dog

        ld      STACKEND,#STACKTOP        ; start at th  top of the stack
SETSTACKLOOP:
        ld      @STACKEND,#01H                  ; set the value for the stack vector
        dec     STACKEND                        ; next address
        cp      STACKEND,#STACKEND        ; test for the last address
        jr      nz,SETSTACKLOOP                 ; loop till done

CLEARDONE:


;       ld      STATE,#06                       ; set the state to stop
;       ld      BSTATE,#06                      ;
;       ld      OnePass,STATE             ; Set the one-shot
        ld      STATUS,#CHARGE                  ; set start to charge
        ld      SWITCH_DELAY,#CMD_DEL_EX  ; set the delay time to cmd
        ld      LIGHT_TIMER_HI,#USA_LIGHT_HI    ; set the light period
        ld      LIGHT_TIMER_LO,#USA_LIGHT_LO    ; for the 4.5 min timer
        ld      RPMONES,#244              ; set the hold off
        srp     #LEARNEE_GRP              ;
        ld      learndb,#0FFH             ; set the learn debouncer
        ld      zzwin,learndb             ; turn off the learning
        ld      CMD_DEB,learndb                 ; in case of shorted switches
        ld      BCMD_DEB,learndb                ; in case of shorted switches
        ld      VAC_DEB,learndb                 ;
        ld      LIGHT_DEB,learndb         ;
        ld      ERASET,learndb                  ; set the erase timer
        ld      learnt,learndb                  ; set the learn timer
        ld      RTO,learndb                     ; set the radio time out
        ld      AUXLEARNSW,learndb        ; turn off the aux learn switch
        ld      RRTO,learndb              ; set the radio timer

;*****************************************************************
; STACK INITILIZATION
;*****************************************************************
        clr     254
        ld      255,#238                        ; set the start of the stack
        .IF     TwoThirtyThree
        .ELSE
        clr     P1
        .ENDIF


;*****************************************************************
; TIMER INITILIZATION
;*****************************************************************

        ld      PRE0,#00000101B                 ; set the prescaler to  /1 for 4MHz
        ld      PRE1,#00010011B                 ; set the prescaler to  /4 for 4MHz
        clr     T0                              ; set the counter to count FF through 0
        ld      T1,#RsPerHalf             ; set the period to rs232 period for start bit sample
        ld      TMR,#00001111B                  ; turn on the timers

;*****************************************************************
; PORT INITILIZATION
;*****************************************************************
        ld      P0,#P01S_INIT             ; RESET all ports
        ld      P2,#P2S_INIT              ;
        ld      P3,#P3S_INIT              ;
        ld      P01M,#P01M_INIT                 ; set mode p00-p03 out p04-p07in
        ld      P3M,#P3M_INIT                   ; set port3 p30-p33 input analog mode
                                                ; p34-p37 outputs
        ld      P2M_SHADOW,#P2M_INIT            ; Shadow P2M for read ability
        ld      P2M,#P2M_INIT             ; set port 2 mode

        .IF     TwoThirtyThree
        .ELSE
        clr     P1
        .ENDIF

;*********************************************************************
; READ THE MEMORY 2X AND GET THE VACFLAG
;*********************************************************************


        ld      SKIPRADIO,#NOEECOMM       ;
        ld      ADDRESS,#VACATIONADDR             ; set non vol address to the VAC flag
        call    READMEMORY                        ; read the value 2X 1X INIT 2ND read
        call    READMEMORY                        ; read the value
        ld      VACFLAG,MTEMPH                    ; save into volital

WakeUpLimits:

        ld      ADDRESS, #UPLIMADDR       ; Read the up and down limits into memory
        call    READMEMORY                        ;
        ld      UP_LIMIT_HI, MTEMPH       ;
        ld      UP_LIMIT_LO, MTEMPL       ;
        ld      ADDRESS, #DNLIMADDR       ;
        call    READMEMORY                        ;
        ld      DN_LIMIT_HI, MTEMPH       ;
        ld      DN_LIMIT_LO, MTEMPL       ;
        WDT                                       ; Kick the dog

WakeUpState:
        ld      ADDRESS, #LASTSTATEADDR           ; Read the previous operating state into memory
        call    READMEMORY                        ;
        ld      STATE, MTEMPL             ; Load the state
        ld      PassCounter, MTEMPH       ; Load the pass point counter
        cp      STATE, #UP_POSITION       ; If at up limit, set position
        jr      z, WakeUpLimit                    ;
        cp      STATE, #DN_POSITION       ; If at down limit, set position
        jr      z, WakeDnLimit                    ;

WakeUpLost:
        ld      STATE, #STOP              ; Set state as stopped in mid travel
        ld      POSITION_HI, #07FH        ; Set position as lost
        ld      POSITION_LO, #080H        ;
        jr      GotWakeUp                         ;

WakeUpLimit:
        ld      POSITION_HI, UP_LIMIT_HI  ; Set position as at the up limit
        ld      POSITION_LO, UP_LIMIT_LO  ;
        jr      GotWakeUp

WakeDnLimit:
        ld      POSITION_HI, DN_LIMIT_HI  ; Set position as at the down limit
        ld      POSITION_LO, DN_LIMIT_LO  ;

GotWakeUp:

        ld      BSTATE, STATE             ; Back up the state and
        ld      OnePass, STATE                    ; clear the one-shot

;*********************************************************************
; SET ROLLING/FIXED MODE FROM NON-VOLATILE MEMORY
;*********************************************************************

        call    SetRadioMode              ; Set the radio mode
        jr      SETINTERRUPTS             ; Continue on

SetRadioMode:

        ld      SKIPRADIO, #NOEECOMM              ; Set skip radio flag
        ld      ADDRESS, #MODEADDR        ; Point to the radio mode flag
        call    READMEMORY                        ; Read the radio mode
        ld      RadioMode, MTEMPL                 ; Set the proper radio mode
        clr     SKIPRADIO                       ; Re-enable the radio
        tm      RadioMode, #ROLL_MASK           ; Do we want rolling numbers
        jr      nz, StartRoll

        call    FixedNums
        ret

StartRoll:

        call    RollNums
        ret

;*******************************************************************
; INITERRUPT INITILIZATION
;*******************************************************************
SETINTERRUPTS:
        ld      IPR,#00011010B                  ; set the priority to timer
        ld      IMR,#ALL_ON_IMR                 ; turn on the interrupt

        .IF     TwoThirtyThree
        ld      IRQ,#01000000B                  ; set the edge clear int
        .ELSE
        ld      IRQ,#00000000b                  ; Set the edge, clear ints
        .ENDIF

;       ei                                      ; enable interrupt

;*******************************************************************
; RESET SYSTEM REG
;*******************************************************************


                .IF     TwoThirtyThree

        ld      RP,#WATCHDOG_GROUP
        ld      smr,#00100010B                  ; reset the xtal / number
        ld      pcon,#01111110B                 ; reset the pcon no comparator output
                                                ; no low emi mode
        clr     RP                              ; Reset the RP

                .ENDIF

        ld      PRE0,#00000101B                 ; set the prescaler to / 1 for 4Mhz
        WDT                                     ; Kick the dog


;*******************************************************************
; MAIN LOOP
;*******************************************************************
MAINLOOP:

        cp      PrevPass, PassCounter                   ;Compare pass point counter to backup
        jr      z, PassPointCurrent             ;If equal, EEPROM is up to date

PassPointChanged:

        ld      SKIPRADIO, #NOEECOMM            ; Disable radio EEPROM communications
        ld      ADDRESS, #LASTSTATEADDR         ; Point to the pass point storage
        call    READMEMORY                      ; Get the current GDO state
        di                                      ; Lock in the pass point state
        ld      MTEMPH, PassCounter     ; Store the current pass point state
        ld      PrevPass, PassCounter           ; Clear the one-shot
        ei                                      ;
        call    WRITEMEMORY                     ; Write it back to the EEPROM
        clr     SKIPRADIO                       ;

PassPointCurrent:
;
;4-22-97
```

```
        CP      EnableWorkLight,#10000000B ;is the debouncer set? if so write and
                                           ;give feedback
        JR      NE,LightOpen
        TM      p0,#LIGHT_ON
        JR      NZ,GetRidOfIt
        LD      MTEMPL,#0FFH               ;turn on the IR beam work light function
        LD      MTEMPH,#0FFH
        JR      CommitToMem
GetRidOfIt:
        LD      MTEMPL,#00H                        ;turn off the IR beam work light function
        LD      MTEMPH,#00H
CommitToMem:
        LD      SKIPRADIO,#NOEECOMM        ;write to memory to store if enabled or not
        LD      ADDRESS,#IRLIGHTADDR               ;set address for write
        CALL    WRITEMEMORY
        CLR     SKIPRADIO
        XOR     p0,#WORKLIGHT              ;toggle current state of work light for feedback
        LD      EnableWorkLight,#01100000B
;

LightOpen:
        cp      LIGHT_TIMER_HI,#0FFH               ; if light timer not done test beam break
        jr      nz,TestBeamBreak
        tm      p0,#LIGHT_ON               ; if the light is off test beam break
        jr      nz,LightSkip

TestBeamBreak:
        tm      AOBSF,#10000000b                   ; Test for broken beam
        jr      z,LightSkip                ; if no pulses Staying blocked
                                                   ; else we are intermittent
;4-22-97
        LD      SKIPRADIO,#NOEECOMM        ;Trun off radio interrupt to read from e2
        LD      ADDRESS,#IRLIGHTADDR               ;
        CALL    READMEMORY
        CLR     SKIPRADIO                          ; don't forget to zero the one shot
        CP      MTEMPL,#DISABLED                   ;Does e2 report that IR work light function
        JR      EQ,LightSkip               ;is disabled?  IF so jump over light on and
;
        cp      STATE,#2                           ; test for the up limit
        jr      nz,LightSkip               ; if not goto output the code
        call    SetVarLight                        ; Set worklight to proper time
        or      p0,#LIGHT_ON               ; turn on the light
LightSkip:
;4-22-97
        AND     AOBSF,#01111111B                   ;Clear the one shot,for IR beam
                                                   ;break detect.
;
        cp      HOUR_TIMER_HI, #01CH               ; If an hour has passed,
        jr      ult, NoDecrement                   ; then decrement the
        cp      HOUR_TIMER_LO, #020H               ; temporary password timer
        jr      ult, NoDecrement                   ;

        clr     HOUR_TIMER_HI              ; Reset hour timer
        clr     HOUR_TIMER_LO              ;
        ld      SKIPRADIO, #NOEECOMM               ; Disable radio EE read
        ld      ADDRESS, #DURAT                    ; Load the temporary password
        call    READMEMORY                         ; duration from non-volatile
        cp      MTEMPH, #HOURS                     ; If not in timer mode,
        jr      nz, NoDecrement2                   ; then don't update
        cp      MTEMPL, #00                        ; If timer is not done,
        jr      z, NoDecrement2                    ; decrement it

        dec     MTEMPL                     ; Update the number of hours
        call    WRITEMEMORY                        ;

NoDecrement:

        tm      AOBSF, #01000000b                  ; If the poll radio mode flag is
        jr      z, NoDecrement2                    ; set, poll the radio mode
        call    S tRadioMode                    ; Set the radio mode
        and     AOBSF, #10111111b                        ; Cl ar the flag

NoDecrement2:

        clr     SKIPRADIO                                ; Re-enable radio reads
        and     AOBSF,#00100011b                         ; Clear the single break flag
        clr     DOG2                                     ; clear the second watchdog
        ld      P01M,#P01M_INIT                          ; set mode p00-p03 out p04-p07in
        ld      P3M,#P3M_INIT                            ; set port3 p30-p33 input analog mode
                                                         ; p34-p37 outputs
        or      P2M_SHADOW,#P2M_ALLINS                   ; Refresh all the P2M pins which have are
        and     P2M_SHADOW,#P2M_ALLOUTS                  ; always the same when we get here
        ld      P2M,P2M_SHADOW                           ; set port 2 mode
        cp      VACCHANGE,#0AAH                          ; test for the vacation change flag
        jr      nz,NOVACCHG                     ; if no change the skip
        cp      VACFLAG,#0FFH                            ; test for in vacation
        jr      z,MCLEARVAC                     ; if in vac clear
        ld      VACFLAG,#0FFH                            ; set vacation
        jr      SETVACCHANGE                    ; set the change
MCLEARVAC:
        clr     VACFLAG                                  ; clear vacation mode
SETVACCHANGE:
        clr     VACCHANGE                                ; one shot
        ld      SKIPRADIO,#NOEECOMM                      ; set skip flag
        ld      ADDRESS,#VACATIONADDR                    ; set the non vol address to the VAC flag
        ld      MTEMPH,VACFLAG                           ; store the vacation flag
        ld      MTEMPL,VACFLAG                           ;
        call    WRITEMEMORY                     ; write the value
        clr     SKIPRADIO                                ; clear skip flag
NOVACCHG:
        cp      STACKFLAG,#0FFH                          ; test for the change flag
        jr      nz,NOCHANGEST                            ; if no change skip updating

        cp      L_A_C, #070H                    ; If we're in learn mode
        jr      uge, SkipReadLimits             ; then don't refresh the limits!

        cp      STATE, #UP_DIRECTION                     ; If we are going to travel up
        jr      z, ReadUpLimit                           ; then read the up limit

        cp      STATE, #DN_DIRECTION                     ; If we are going to travel down
        jr      z, ReadDnLimit                           ; then read the down limit

        jr      SkipReadLimits                           ; No limit on this travel...

ReadUpLimit:

        ld      SKIPRADIO, #NOEECOMM                     ; Skip radio EEPROM reads
        ld      ADDRESS, #UPLIMADDR             ; Read the up limit
        call    READMEMORY                               ;
        di                                               ;
        ld      UP_LIMIT_HI, MTEMPH             ;
        ld      UP_LIMIT_LO, MTEMPL             ;
        clr     FirstRun                                 ; Calculate the highest possible value for pass count
        add     MTEMPL, #10                              ; Bias back by 1" to provide margin of error
        adc     MTEMPH, #00                              ;
CalcMaxLoop:
        inc     FirstRun                                 ;
        add     MTEMPL, #LOW(PPOINTPULSES) ;
        adc     MTEMPH, #HIGH(PPOINTPULSES)              ;
        jr      nc, CalcMaxLoop                          ; Count pass points until value goes positive
GotMaxPPoint:
        ei                                               ;
        clr     SKIPRADIO                                ;
        tm      PassCounter, #01000000b                  ; Test for a negative pass point counter
        jr      z, CounterGood1                          ; If not, no lower bounds check needed
        cp      DN_LIMIT_HI, #HIGH(PPOINTPULSES - 35)    ; If the down limit is low enough,
        jr      ugt, CounterIsNeg1              ; then the counter can be negative
        jr      ult, ClearCount                     ; Else, it should be z ro
        cp      DN_LIMIT_LO, #LOW(PPOINTPULSES - 35)
        jr      uge, CounterIsNeg1          ;
ClearCount:
        and     PassCounter, #10000000b             ; Reset the pass point counter to zero
        jr      CounterGood1                ;
CounterIsNeg1:
        or      PassCounter, #01111111b             ; Set the pass point counter to -1
CounterGood1:
        cp      UP_LIMIT_HI, #0FFH          ; Test to make sure up limit is at a
        jr      nz, TestUpLimit2                    ; a learned and legal value
        cp      UP_LIMIT_LO, #0FFH          ;
        jr      z, LimitIsBad               ;
        jr      LimitsAreDone               ;
TestUpLimit2:
        cp      UP_LIMIT_HI, #0D0H          ; Look for up limit set to illegal value
        jr      ule, LimitIsBad                     ; If so, set the limit fault
        jr      LimitsAreDone               ;

ReadDnLimit:

        ld      SKIPRADIO, #NOEECOMM                ; Skip radio EEPROM reads
        ld      ADDRESS, #DNLIMADDR         ; Read the down limit
        call    READMEMORY                          ;
        di                                          ;
        ld      DN_LIMIT_HI, MTEMPH         ;
        ld      DN_LIMIT_LO, MTEMPL         ;
        ei                                          ;
        clr     SKIPRADIO                           ;
        cp      DN_LIMIT_HI, #00H                   ; Test to make sure down limit is at a
        jr      nz, TestDownLimit2          ; a learned and legal value
        cp      DN_LIMIT_LO, #00H                   ;
        jr      z, LimitIsBad               ;
        jr      LimitsAreDone               ;
TestDownLimit2:
        cp      DN_LIMIT_HI, #020H          ; Look for down limit set to illegal value
        jr      ult, LimitsAreDone          ; If not, proceed as normal
LimitIsBad:
        ld      FAULTCODE, #7               ; Set the "no limits" fault
        call    SET_STOP_STATE                      ; Stop the GDO
        jr      LimitsAreDone               ;

SkipReadLimits:
LimitsAreDone:

        ld      SKIPRADIO, #NOEECOMM                ; Turn off the radio read
        ld      ADDRESS, #LASTSTATEADDR             ; Write the current state and pass count
        call    READMEMORY                          ;
;       ld      MTEMPH, PassCounter         ; DON'T update the pass point here!
        ld      MTEMPL, STATE               ;
        call    WRITEMEMORY                         ;
        clr     SKIPRADIO                           ;

        ld      OnePass, STATE                      ; Clear the one-shot

        cp      L_A_C, #077H                ; Test for successful learn cycle
        jr      nz, DontWriteLimits         ; If not, skip writing limits
WriteNewLimits:
        cp      STATE, #STOP                ;
        jr      nz, WriteUpLimit                    ;
        cp      LIM_TEST_HI, #00                    ; Test for (force) stop within 0.5" of
        jr      nz, WriteUpLimit                    ; the original up limit position
        cp      LIM_TEST_LO, #06                    ;
        jr      ugt, WriteUpLimit                   ;
BackOffUpLimit:                                     ;
        add     UP_LIMIT_LO, #06                    ; Back off the up limit by 0.5"
        adc     UP_LIMIT_HI, #00                    ;
WriteUpLimit:
        ld      SKIPRADIO, #NOEECOMM                ; Skip radio EEPROM reads
```

```
     ld   ADDRESS, #UPLIMADDR      ; Read the up limit
     di                                 ;
     ld   MTEMPH, UP_LIMIT_HI      ;
     ld   MTEMPL, UP_LIMIT_LO      ;
     ei                                 ;
     call WRITEMEMORY                   ;
WriteDnLimit:
     ld   ADDRESS, #DNLIMADDR      ; Read the up limit
     di                                 ;
     ld   MTEMPH, DN_LIMIT_HI      ;
     ld   MTEMPL, DN_LIMIT_LO      ;
     ei                                 ;
     call WRITEMEMORY                   ;
WritePassCount:
     ld   ADDRESS, #LASTSTATEADDR       ; Write the current state and pass count
     ld   MTEMPH, PassCounter      ; Update the pass point
     ld   MTEMPL, STATE            ;
     call WRITEMEMORY                   ;
     clr  SKIPRADIO                     ;
     clr  L_A_C                         ; Leave the learn mode
     or   ledport,#ledh            ; turn off the LED for program mode

DontWriteLimits:

     srp  #LEARNEE_GRP             ; set the register pointer
     clr  STACKFLAG                     ; clear the flag
     ld   SKIPRADIO,#NOEECOMM           ; set skip flag
     ld   address,#CYCCOUNT             ; set the non vol address to the cycle c
     call READMEMORY                    ; read the value
     inc  mtempl                   ; increase the counter lower byte
     jr   nz,COUNTER1DONE               ;
     inc  mtemph                        ; increase the counter high byte
     jr   nz,COUNTER2DONE               ;
     call WRITEMEMORY              ; store the value
     inc  address                       ; get the next bytes
     call READMEMORY                    ; read the data
     inc  mtempl                   ; increase the counter low byte
     jr   nz,COUNTER2DONE               ;
     inc  mtemph                   ; increase the vounter high byte
COUNTER2DONE:
     call WRITEMEMORY              ; save the value
     ld   address,#CYCCOUNT             ;
     call READMEMORY                    ; read the data

     and  mtemph,#00001111B        ; find the force address
     or   mtemph,#30H              ;
     ld   ADDRESS,MTEMPH                ; set the address
     ld   mtempl,DNFORCE                ; read the forces
     ld   mtemph,UPFORCE                ;
     call WRITEMEMORY              ; write the value
     jr   CDONE                    ; done set the back trace
COUNTER1DONE:
     call WRITEMEMORY              ; got the new address
CDONE:
     clr  SKIPRADIO                     ; clear skip flag

NOCHANGEST:
     call LEARN                         ; do the learn switch
     di
     cp   BRPM_COUNT,RPM_COUNT
     jr   z,TESTRPM
RESET:
     jp   START
TESTRPM:
     cp   BRPM_TIME_OUT,RPM_TIME_OUT
     jr   nz,RESET
     cp   BFORCE_IGNORE,FORCE_IGNORE
     jr   nz,RESET
     ei
```

```
        di
        cp      BAUTO_DELAY,AUTO_DELAY
        jr      nz,RESET
        cp      BCMD_DEB,CMD_DEB
        jr      nz,RESET
        cp      BSTATE,STATE
        jr      nz,RESET
        ei
TESTRS232:
        SRP     #TIMER_GROUP
        tcm     RS_COUNTER, #00001111B              ; If we are at the end of a word,
        jp      nz, SKIPRS232                   ; then handle the RS232 word

        cp      rscommand,#'V'                      ;
        jp      ugt,ClearRS232                      ;
        cp      rscommand,#'0'                      ; test for in range
        jp      ult,ClearRS232                      ; if out of range skip
        cp      rscommand,#'<'                      ; If we are reading
        jr      nz,NotRs3C                          ; go straight there
        call    GotRs3C                             ;
        jp      SKIPRS232                           ;
NotRs3C:
        cp      rscommand,#'>'                      ; If we are writing EEPROM
        jr      nz,NotRs3E                          ; go straight there
        call    GotRs3E                             ;
        jp      SKIPRS232                           ;
NotRs3E:
        ld      rs_temp_hi,#HIGH (RS232JumpTable-(3*'0'))      ; address pointer to table
        ld      rs_temp_lo,#LOW (RS232JumpTable-(3*'0'))       ; Offset for ASCII adjust

        add     rs_temp_lo,rscommand                ; look up the jump 3x
        adc     rs_temp_hi,#00                      ;
        add     rs_temp_lo,rscommand                ; look up the jump 3x
        adc     rs_temp_hi,#00                      ;
        add     rs_temp_lo,rscommand                ; look up the jump 3x
        adc     rs_temp_hi,#00                      ;
        call    @rs_temp                            ; call this address
        jp      SKIPRS232                           ; done

RS232JumpTable:
        jp      GotRs30
        jp      GotRs31
        jp      GotRs32
        jp      GotRs33
        jp      GotRs34
        jp      GotRs35
        jp      GotRs36
        jp      GotRs37
        jp      GotRs38
        jp      GotRs39
        jp      GotRs3A
        jp      GotRs3B
        jp      GotRs3C
        jp      GotRs3D
        jp      GotRs3E
        jp      GotRs3F
        jp      GotRs40
        jp      GotRs41
        jp      GotRs42
        jp      GotRs43
        jp      GotRs44
        jp      GotRs45
        jp      GotRs46
        jp      GotRs47
        jp      GotRs48
        jp      GotRs49
        jp      GotRs4A
        jp      GotRs4B
        jp      GotRs4C
        jp      GotRs4D
        jp      GotRs4E
        jp      GotRs4F
        jp      GotRs50
        jp      GotRs51
        jp      GotRs52
        jp      GotRs53
        jp      GotRs54
        jp      GotRs55
        jp      GotRs56

ClearRS232:

        and     RS_COUNTER, #11110000b                  ; Clear the RS232 state

SKIPRS232:

UpdateForceAndSpeed:

        ; Update the UP force from the look-up table

        srp     #FORCE_GROUP                    ; Point to the proper registers
        ld      force_add_hi, #HIGH(force_table) ; Fetch the proper unscaled
        ld      force_add_lo, #LOW(force_table)  ; value from the ROM table
        di                                              ;
        add     force_add_lo, upforce                   ; Offset to point to the
        adc     force_add_hi, #00                       ; proper place in the table
        add     force_add_lo, upforce                   ; x2
        adc     force_add_hi, #00                       ;
        add     force_add_lo, upforce                   ; x3 (three bytes wide)
        adc     force_add_hi, #00                       ;
        ei                                              ;

        ldc     force_temp_of, @force_add       ; Fetch the ROM bytes
        incw    force_add                               ;
        ldc     force_temp_hi, @force_add       ;
        incw    force_add                               ;
        ldc     force_temp_lo, @force_add       ;

        ld      Divisor, PowerLevel             ; Divide by our current force level
        call    ScaleTheSpeed                   ; Scale to get our proper force number

        di                                              ; Update the force registers
        ld      UP_FORCE_HI, force_temp_hi      ;
        ld      UP_FORCE_LO, force_temp_lo      ;
        ei                                              ;

        ; Update the DOWN force from the look-up table

        ld      force_add_hi, #HIGH(force_table) ; Fetch the proper unscaled
        ld      force_add_lo, #LOW(force_table)  ; value from the ROM table
        di                                              ;
        add     force_add_lo, dnforce                   ; Offset to point to the
        adc     force_add_hi, #00                       ; proper place in the table
        add     force_add_lo, dnforce                   ; x2
        adc     force_add_hi, #00                       ;
        add     force_add_lo, dnforce                   ; x3 (three bytes wide)
        adc     force_add_hi, #00                       ;
        ei                                              ;

        ldc     force_temp_of, @force_add       ; Fetch the ROM bytes
        incw    force_add                               ;
        ldc     force_temp_hi, @force_add       ;
        incw    force_add                               ;
        ldc     force_temp_lo, @force_add       ;

        ld      Divisor, PowerLevel             ; Divide by our current force level
        call    ScaleTheSpeed                   ; Scale to get our proper force number
```

```
    di                               ; Update the force registers
    ld  DN_FORCE_HI, force_temp_hi   ;
    ld  DN_FORCE_LO, force_temp_lo   ;
    ei                               ;

    ; Scale the minimum speed based on force setting
    cp  STATE, #DN_DIRECTION         ; If we're traveling down,
    jr  z, SetDownMinSpeed       ; then use the down force pot for min. speed
SetUpMinSpeed:
    di                               ; Disable interrupts during update
    ld  MinSpeed, UPFORCE            ; Scale up force pot
    jr  MinSpeedMath             ;
SetDownMinSpeed:
    di                               ;
    ld  MinSpeed, DNFORCE            ; Scale down force pot
MinSpeedMath:
    sub MinSpeed, #24            ;   pot level - 24
    jr  nc, UpStep2                  ;   truncate off the negative number
    clr MinSpeed                     ;
UpStep2:
    rcf                              ;   Divide by four
    rrc MinSpeed                     ;
    rcf                              ;
    rrc MinSpeed                     ;
    add MinSpeed, #4             ;   Add four to find the minimum speed
    cp  MinSpeed, #12            ;   Perform bounds check on minimum speed.
    jr  ule, MinSpeedOkay            ;   Truncate if necessary
    ld  MinSpeed, #12            ;
MinSpeedOkay:
    ei                               ;   Re-enable interrupts

    ; Make sure the worklight is at the proper time on power-up

    cp  LinePer, #36             ; Test for a 50 Hz system
    jr  ult, TestRadioDeadTime       ; if not, we don't have a problem
    cp  LIGHT_TIMER_HI, #0FFH        ; If the light timer is running
    jr  z, TestRadioDeadTime         ; and it is greater than
    cp  LIGHT_TIMER_HI, #EURO_LIGHT_HI   ; the European time, fix it
    jr  ule, TestRadioDeadTime       ;
    call    SetVarLight              ;

TestRadioDeadTime:

    cp  R_DEAD_TIME,#25          ; test for too long dead
    jp  nz,MAINLOOP          ; if not loop
    clr RadioC                   ; clear the radio counter
    clr RFlag                    ; clear the radio flag
    jp  MAINLOOP                 ; loop forever

;-----------------------------------------------------------------
;   Speed scaling (i.e. Division) routine
;-----------------------------------------------------------------

ScaleTheSpeed:

    clr TestReg
    ld  loopreg, #24             ; Loop for all 24 bits
DivideLoop:
    rcf                              ; Rotate the next bit into
    rlc force_temp_lo            ; the test field
    rlc force_temp_hi            ;
    rlc force_temp_of            ;
    rlc TestReg                      ;
    cp  TestReg, Divisor             ; Test to see if we can subtract
    jr  ult, BitIsDone               ; If we can't, we're all done
    sub TestReg, Divisor             ; Subtract the divisor
    or  force_temp_lo, #00000001b    ; Set the LSB to mark the subtract
BitIsDone:
    djnz    loopreg, DivideLoop      ; Loop for all bits
```

```
DivideDone:
       ; Make sure the result is under our 500 ms limit
       cp     force_temp_of, #00              ; Overflow byte must be zero
       jr     nz, ScaleDown                   ;
       cp     force_temp_hi, #0F4H                   ;
       jr     ugt,ScaleDown                   ;
       jr     ult, DivideIsGood                      ; If we're less, then we're okay
       cp     force_temp_lo, #024H                   ; Test low byte
       jr     ugt,ScaleDown                   ; if low byte is okay,
DivideIsGood:
       ret                                           ; Number is good

ScaleDown:
       ld     force_temp_hi, #0F4H                   ; Overflow is never used anyway
       ld     force_temp_lo, #024H                   ;
       ret


;*********************************************************************
; RS232 SUBROUTINES
;*********************************************************************
; "0"
; Set Command Switch
GotRs30:
       ld     LAST_CMD,#0AAH                         ; set the last command as rs wall cmd

       call   CmdSet                          ; set the command switch
       jp     NoPos

; "1"
; Clear Command Switch
GotRs31:
       call   CmdRel                          ; release the command switch
       jp     NoPos

; "2"
; Set Worklight Switch
GotRs32:
       call   LightSet                               ; set the light switch
       jp     NoPos

; "3"
; Clear Worklight Switch
GotRs33:
       clr    LIGHT_DEB                              ; Release the light switch
       jp     NoPos

; "4"
; Set Vacation Switch
GotRs34:
       call   VacSet                          ; Set the vacation switch
       jp     NoPos

; "5"
; Cl ar Vacation Switch
GotRs35:
       clr    VAC_DEB                                ; release the vacation switch
       jp     NoPos

; "6"
; Set smart switch
GotRs36:
       call   SmartSet
       jp     NoPos


; "7"
; Clear Smart switch set
GotRs37:
       call  SmartRelease
       jp    NoPos

; "8"
; Return Present state and reason for that state
GotRs38:
       ld    RS232DAT,STATE
       or    RS232DAT,STACKREASON
       jp    LastPos

; "9"
; Return Force Adder and Fault
GotRs39:
       ld    RS232DAT,FAULTCODE               ; insert the fault code.
       jp    LastPos

; ":"
; Status Bits
GotRs3A:
       clr   RS232DAT                                ; Reset data
       tm    P2, #01000000b                          ; Check the strap
       jr    z, LookForBlink                         ; If none, next check
       or    RS232DAT, #00000001b                    ; Set flag for strap high

LookForBlink:

       call  LookForFlasher                          ;
       tm    P2, #BLINK_PIN                          ; If flasher is present,
       jr    nz, ReadLight                    ;
       or    RS232DAT,#00000010b              ; then indicate it

ReadLight:

       tm    P0,#00000010B                    ; read the light
       jr    z,C3ADone
       or    RS232DAT,#00000100b
C3ADone:

       cp    CodeFlag, #REGLEARN              ; Test for being in a learn mode
       jr    ult, LookForPass                        ; If so, set the bit
       or    RS232DAT,#00010000b              ;

LookForPass:

       tm    PassCounter,#01111111b                  ; Check for above pass point
       jr    z, LookForProt                          ; If so, set the bit
       tcm   PassCounter,#01111111b                  ;
       jr    z, LookForProt
       or    RS232DAT,#00100000b              ;

LookForProt:

       tm    ACBSF, #10000000b                       ; Check for protector break/block
       jr    nz, LookForVac                          ; If blocked, don't set the flag
       or    RS232DAT,#01000000b              ; Set flag for protector signal good

LookForVac:

       cp    VACFLAG,#00B                     ; test for the vacation mode
       jp    nz,LastPos
       or    RS232DAT,#00001000b
       jp    LastPos

; ";"
; Return L_A_C
GotRs3B:
       ld    RS232DAT,L_A_C                          ; read the L_A_C
       jp    LastPos
```

```
; "<"
; Read a word of data from an EEPROM address input by the user
GotRs3C:
        cp      RS_COUNTER, #010H                    ; If we have only received the
        jr      ult, FirstByte                       ; first word, wait for more
        cp      RS_COUNTER, #080H                    ; If we are outputting,
        jr      ugt, OutputSecond                    ; output the second byte

SecondByte:

        ld      SKIPRADIO, #0FFH                     ; Read the memory at the specified
        ld      ADDRESS, RS232DAT                    ; address
        call    READMEMORY                           ;
        ld      RS232DAT, MTEMPH                     ; Store into temporary registers
        ld      RS_TEMP_LO, MTEMPL              ;
        clr     SKIPRADIO                            ;
        jp      MidPos                          ;

OutputSecond:

        ld      RS232DAT, RS_TEMP_LO                 ; Output the second byte of the read
        jp      LastPos                              ;

FirstByte:

        inc     RS_COUNTER                           ; Set to receive second word
        ret                                          ;

; "="
; Exit learn limits mode
GotRs3D:
        cp      L_A_C, #00                           ; If not in learn mode,
        jp      z, NoPos                             ; then don't touch the learn LED
        clr     L_A_C                                ; Reset the learn limits state machine
        or      ledport,#ledh                   ; turn off the LED for program mode
        jp      NoPos                                ;

; ">"
; Write a word of data to the address input by the user
GotRs3E:

        cp      RS_COUNTER, #01FH                    ;
        jr      z, SecondByteW                       ;
        cp      RS_COUNTER, #02FH                    ;
        jr      z, ThirdByteW                   ;
        cp      RS_COUNTER, #03FH                    ;
        jr      z, FourthByteW                       ;

FirstByteW:
DataDone:

        inc     RS_COUNTER                           ; Set to receive next byte
        ret                                          ;

SecondByteW:

        ld      RS_TEMP_HI, RS232DAT                 ; Store the address
        jr      DataDone                             ;

ThirdByteW:

        ld      RS_TEMP_LO, RS232DAT                 ; Store the high byte
        jr      DataDone                             ;

FourthByteW:

        cp      RS_TEMP_HI, #03FH                    ; Test for illegal address
        jr      ugt, FailedWrite                     ; If so, don't write
        ld      SKIPRADIO, #0FFH                ; Turn off radio reads
        ld      ADDRESS, RS_TEMP_HI           ; Load the address
        ld      MTEMPH, RS_TEMP_LO            ; and the data for the
        ld      MTEMPL, RS232DAT                ; EEPROM write
        call    WRITEMEMORY                     ;
        clr     SKIPRADIO                       ; Re-enable radio reads
        ld      RS232DAT, #00H                  ; Flag write okay
        jp      LastPos                         ;

FailedWrite:

        ld      RS232DAT, #0FFH                 ; Flag bad write
        jp      LastPos

; "?"
; Suspend all communication for 30 seconds
GotRs3F:
        clr     RSCOMMAND                       ; Throw out any command currently
                                                ; running
        jp      NoPos                           ; Ignore all RS232 data

; "@"
; Force Up State
GotRs40:
        cp      STATE, #DN_DIRECTION            ; If traveling down, make sure that
        jr      z, dontup                       ; the door autoreverses first
        cp      STATE, #AUTO_REV                ; If the door is autoreversing or
        jp      z, NoPos                        ; at the up limit, don't let the
        cp      STATE, #UP_POSITION           ; up direction state be set
        jp      z, NoPos                        ;
        ld      REASON, #00H                  ; Set the reason as command
        call    SET_UP_DIR_STATE
        jp      NoPos
dontup:
        ld      REASON, #00H                  ; Set the reason as command
        call    SET_AREV_STATE                  ; Autoreverse the door
        jp      NoPos                           ;

; "A"
; Force Down State
GotRs41:
        cp      STATE,#5h                       ; test for the down position
        jp      z,NoPos                         ;

        clr     REASON                        ; Set the reason as command
        call    SET_DN_DIR_STATE
        jp      NoPos

; "B"
; Force Stop State
GotRs42:
        clr     REASON                        ; Set the reason as command
        call    SET_STOP_STATE
        jp      NoPos

; "C"
; Force Up Limit State
GotRs43:
        clr     REASON                        ; Set the reason as command
        call    SET_UP_POS_STATE
        jp      NoPos

; "D"
; Force Down Limit State
GotRs44:
        clr     REASON                        ; Set the reason as command
        call    SET_DN_POS_STATE
        jp      NoPos
```

```
; "E"
; Return min. force during travel
GotRs45:
;       ld      RS232DAT,MIN_RPM_HI                 ; Return high and low
;       cp      RS_COUNTER,#090h                          ; bytes of min. force read
;       jp      ult,MidPos                                ;
;       ld      RS232DAT,MIN_RPM_LO                 ;
        jp      LastPos                                   ;


; "F"
; Leave RS232 mode -- go back to scanning for wall control switches
GotRs46:

        clr     RsMode                              ; Exit the rs232 mode
        ld      STATUS, #CHARGE                           ; Scan for switches again
        clr     RS_COUNTER                                ; Wait for input again
        ld      rscommand,#0FFH                           ; turn off command
        ret


; "G"
; (No Function)
;
GotRs47:
        jp      NoPos

; "H"
; 45 Second search for pass point the setup for the door
;
GotRs48:
        ld      SKIPRADIO, #0FFH                          ; Disable radio EEPROM reads / writes
        ld      MTEMPH, #0FFH                       ; Erase the up limit and down limit
        ld      MTEMPL, #0FFH                       ; in EEPROM memory
        ld      ADDRESS, #UPLIMADDR                 ;
        call    WRITEMEMORY                               ;
        ld      ADDRESS, #DNLIMADDR                 ;
        call    WRITEMEMORY                               ;
        ld      UP_LIMIT_HI, #HIGH(SetupPos)              ; Set the door to travel
        ld      UP_LIMIT_LO, #LOW(SetupPos)               ; to the setup position
        ld      POSITION_HI, #040H                  ; Set the current position to unknown
        and     PassCounter, #10000000b                   ; Reset to activate on first pass point seen
        call    SET_UP_DIR_STATE                          ; Force the door to travel
        ld      OnePass, STATE                            ; without a limit refresh
        jp      NoPos

; "I"
; Return radio drop-out timer
GotRs49:
        clr     RS232DAT                                  ; Initially say no radio on
        cp      RTO,#RDROPTIME                            ; If there's no radio on,
        jp      uge, LastPos                        ; then broadcast that
        com     RS232DAT                                  ; Set data to FF
        jp      LastPos

; "J"
; Return current position
GotRs4A:
        ld      RS232DAT,POSITION_HI
        cp      RS_COUNTER,#090H                          ; Test for no words out yet
        jp      ult,MidPos                                ; If not, transmit high byte
        ld      RS232DAT,POSITION_LO
        jp      LastPos


; "K"
; Set radio Received
GotRs4B:
        cp      L_A_C, #070H        ; If we were positioning the up limit,
        jr      ult, NormalRSRadio  ; then start the learn cycle
        jr      z, FirstRSLearn             ;
        cp      L_A_C, #071H        ; If we had an  rror,
        jp      nz, NoPos                   ; re-learn, otherwise ignore
ReLearnRS:
        ld      L_A_C, #072H        ; Set the re-learn state
        call    SET_UP_DIR_STATE            ;
        jp      NoPos                       ;
FirstRSLearn:
        ld      L_A_C, #073H        ; Set the learn state
        call    SET_UP_POS_STATE            ; Start from the "up limit"
        jp      NoPos                       ;
NormalRSRadio:
        clr     LAST_CMD                    ; mark the last command as radio
        ld      RADIO_CMD,#0AAH             ; set the radio command
        jp      NoPos                       ; return

; "L"
; Direct-connect sensitivity test -- toggle worklight for any code
GotRs4C:
;       clr     RTO                                         ; Reset the drop-out timer
;       ld      CodeFlag, #SENS_TEST                        ; Set the flag to test sensitivity
        jp      NoPos

; "M"
GotRs4D:
        jp      NoPos

; "N"
; If we are within the first 4 seconds and RS232 mode is not yet enabled,
; then echo the nybble on P30 - P33 on all other nybbles
; (A.K.A. The 6800 test)
GotRs4E:

        cp      SDISABLE, #32                       ; If the 4 second init timer
        jp      ult, ExitNoTest                             ; is done, don't do the test

        di                                                  ; Shut down all other GDO operations
        ld      COUNT_HI, #002H                             ; Set up to loop for 512 iterations,
        clr     COUNT_LO                                    ; totaling 13.056 milliseconds
        ld      P01M, #00000100b                            ; Set all possible pins of micro.
        ld      P2M, #00000000b                             ; to outputs for testing
        ld      P3M, #00000001b                             ;
        WDT                                                 ; Kick the dog

TimingLoop:

        clr     REGTEMP                                     ; Create a byte of identical nybbles
        ld      REGTEMP2, P3                        ; from P30 - P33 to write to all ports
        and     REGTEMP2, #00001111b                        ;
        or      REGTEMP, REGTEMP2                           ;
        swap    REGTEMP2                                    ;
        or      REGTEMP, REGTEMP2                           ;
        ld      P0, REGTEMP                                 ; Echo the nybble to all ports
        ld      P2, REGTEMP                                 ;
        ld      P3, REGTEMP                                 ;
        decw    COUNT                                       ; Loop for 512 iterations
        jr      nz, TimingLoop                              ;
        jp      START                                       ; When done, reset the system


; "O"
;       Return max. force during travel
;
GotRs4F:
;       ld      RS232DAT,P32_MAX_HI                 ; Return high and low
;       cp      RS_COUNTER,#090h                            ; bytes of max. force read
;       jp      ult,MidPos                                  ;
;       ld      RS232DAT,P32_MAX_LO                     ;
        jp      LastPos                                 ;


; "P"
; Return the measured temperature range
GotRs50:

        jr      NoPos                                   ;

; "Q"
; Return address of last memory matching
; radio code received
GotRs51:

;       ld      RS232DAT, RTEMP                         ; Send back the last matching address
        jr      LastPos                                 ;


; "R"
; Set Rs232 mode -- No ultra board present
; Return Version
GotRs52:
;       clr     UltraBrd                                ; Clear flag for ultra board present
SetIntoRs232:
        ld      RS232DAT,#VERSIONNUM                    ; Initially return the version
        cp      RsMode,#00                              ; If this is the first time we're
        jr      ugt, LockedInNoCR                       ; locking RS232, signal it
        ld      RS232DAT,#0BBH                          ; Return a flag for initial RS232 lock

LockedInNoCR:
;       ld      RsMode,#32
        jr      LastPos

; "S"
; Set Rs232 mode -- Ultra board present
; Return Version
GotRs53:

        jr      NoPos

; "T"
; Range test -- toggle worklight whenever a good memory-matching code
; is received
GotRs54:

;       clr     RTO                                     ; Reset the drop-out timer
;       ld      CodeFlag, #RANGETEST                    ; Set the flag to test sensitivity
        jr      NoPos

; "U"
; (No Function)
GotRs55:

        jr      NoPos

; "V"
; Return current values of up and down force pots
GotRs56:

        ld      RS232DAT,UPFORCE                        ; Return values of up and down
        cp      RS_COUNTER,#090h                        ; force pots.
        jp      ult,MidPos                              ;
        ld      RS232DAT,DNFORCE                        ;
        jr      LastPos

MidPos:
        or      RS_COUNTER, #10000000B                  ; Set the output mode
        inc     RS_COUNTER                              ; Transmit the next byte
        jr      RSDone                          ; exit

LastPos:
        ld      RS_COUNTER, #11110000B                  ; set the start flag for last byte
        ld      rscommand,#0FFH                         ; Cl ar the command
        jr      RSDone                          ; Exit
ExitNoTest:
NoPos:
        clr     RS_COUNTER                              ; Wait for input again
        ld      rscommand,#0FFH                         ; turn off command
RSDone:
        ld      RsMode,#32                              ;
        ld      STATUS, #RSSTATUS                       ; Set the wall control to RS232
        or      P3, #CHARGE_SW                          ; Turn on the pull-ups
        and     P3, #~DIS_SW                    ;
        ret



;***********************************************************************
; Radio interrupt from a edge of the radio signal
;***********************************************************************

RADIO_INT:
        push    RP                              ; save the radio pair
        srp     #RadioGroup                     ; set the register pointer

        ld      rtemph,T0EXT               ; read the upper byte
        ld      rtempl,T0                       ; read the lower byte
        tm      IRQ,#00010000B                  ; test for pending int
        jr      z,RTIMEOK                       ; if not then ok time
        tm      rtempl,#10000000B          ; test for timer reload
        jr      z,RTIMEOK                       ; if not reloaded then ok
        dec     rtemph                     ; if reloaded then dec high for sync
RTIMEOK:
        clr     R_DEAD_TIME                     ; clear the dead time

        .IF     TwoThirtyThree
        and     IMR,#11111110B                  ; turn off the radio interrupt
        .ELSE
        and     IMR,#11111100B                  ; Turn off the radio interrupt
        .ENDIF

        ld      RTimeDH,RTimePH                 ; find the difference
        ld      RTimeDL,RTimePL                 ;
        sub     RTimeDL,rtempl                  ;
        sbc     RTimeDH,rtemph                  ; in past time and the past time in temp
RTIMEDONE:
        tm      P3,#00000100B                   ; test the port for the edge
        jr      nz,ACTIVETIME                   ; if it was the active time then branch
INACTIVETIME:
        cp      RINFILTER,#0FFH                 ; test for active last time
        jr      z,GOINACTIVE               ; if so continue
        jp      RADIO_EXIT                      ; if not the return
GOINACTIVE:

        .IF     TwoThirtyThree
        or      IRQ,#01000000B                  ; set the bit setting direction to pos edge
        .ENDIF

        clr     RINFILTER                       ; set flag to inactive
        ld      rtimeih,RTimeDH                 ; transfer difference to inactive
        ld      rtimeil,RTimeDL                 ;
        ld      RTimePH,rtemph                  ; transfer temp into the past
        ld      RTimePL,rtempl                  ;
;
        CP      radioc,#01H                ;inactive time after sync bit
        JP      NZ,RADIO_EXIT         ;exit if it was not sync
;
```

```
        TM      RadioMode, #ROLL_MASK       ;If in fixed mode,
        JR      z, FixedBlank         ;no number counter exists
        CP      rtimeih,#0AH          ;2.56ms for rolling code mode
        JP      ULT,RADIO_EXIT                ;pulse ok exit as normal
        CLR     radioc                ;if pulse is longer, bogus sync, restart sync search
        jp      RADIO_EXIT                          ; return

FixedBlank:
        CP      rtimeih,#014H         ; test for the max width 5.16ms
        JP      ULT,RADIO_EXIT                ;pulse ok exit as normal
        CLR     radioc                ;if pulse is longer, bogus sync, restart sync search
;
        jp      RADIO_EXIT                          ; return
ACTIVETIME:
        cp      RINFILTER,#00H                      ; test for active last time
        jr      z,GOACTIVE                          ; if so continue
        jr      RADIO_EXIT                          ; if not the return
GOACTIVE:

        .IF     TwoThirtyThree
        and     IRQ,#00111111B                      ; clear bit setting direction to neg edge
        .ENDIF

        ld      RINFILTER,#0FFH                     ;
        ld      rtimeah,RTimeDH                     ; transfer difference to active
        ld      rtimeal,RTimeDL                     ;
        ld      RTimePH,rtemph                      ; transfer temp into the past
        ld      RTimePL,rtempl                      ;
GotBothEdges:
                ei                                  ; enable the interrupts
                cp      radioc,#1                   ; test for the blank timing
                jp      ugt,INSIG                   ; if not then in the middle of signal
        .IF UseSiminor
                jp      z, CheckSiminor             ; Test for a Siminor tx on the first bit
        .ENDIF
                inc     radioc              ; set the counter to the next number
;
                TM      RFlag,#00100000B            ;Has a valid blank time occured
                JR      NZ,BlankSkip
;
                cp      RadioTimeOut,#10            ; test for the min 10 ms blank time
                jr      ult,ClearJump       ; if not then clear the radio
;
                OR      RFlag,#00100000B            ;blank time valid! no need to check
BlankSkip:
                cp      rtimeah,#00h        ; test first the min sync
                jr      z,JustNoise                 ; if high byte 0 then clear the radio
SyncOk:
;
                TM      RadioMode,#ROLL_MASK        ;checking sync pulse width,fix or Roll
                JR      z,Fixedsync
                CP      rtimeah,#09h        ;time for roll 1/2 fixed, 2.3ms
                JR      uge,JustNoise
                JR      SET1
;
Fixedsync:      cp      rtimeah,#012h       ; test for the max time 4.6mS
                jr      uge,JustNoise       ; if not clear

SET1:
                clr     PREVFIX                     ;Clear the previous "fixed" bit
                cp      rtimeah, SyncThresh ; test for 1 or three time units
                jr      uge,SYNC3FLAG               ; set the sync 3 flag
SYNC1FLAG:
                tm      RFlag, #01000000b           ;Was a sync 1 word the last received?
                jr      z, SETADCODE        ;   if not, then this is an A (or D) code

SETBCCODE:

                ld      radio3h, radio1h            ;Store the last sync 1 word
```

```
            ld      radio3l, radio1l
            or      RFlag, #00000110b          ;Set the B/C Code flags
            and     RFlag, #11110111b          ;Clear the A/D Code Flag
            jr      BCCODE
JustNoise:
            CLR     radioc              ;Edge was noise k  p waiting for sync bit
            JP      RADIO_EXIT

SETADCODE:

            or      RFlag, #00001000b

BCCODE:

            or      RFlag,#01000000b           ; set the sync 1 memory flag
            clr     radio1h                    ; clear the memory
            clr     radio1l                    ;
            clr     COUNT1H                    ; clear the memory
            clr     COUNT1L                    ;
            jr      DONESET1                   ; do the 2X
SYNC3FLAG:
            and     RFlag,#10111111b           ; set the sync 3 memory flag
            clr     radio3h                    ; clear the memory
            clr     radio3l                    ;
            clr     COUNT3H                    ; clear the memory
            clr     COUNT3L                    ;
            clr     ID_B                       ; Clear the ID bits
DONESET1:
RADIO_EXIT:
            and     SKIPRADIO, # LOW(~NOINT)   ;Re-enable radio ints
            pop     rp
            iret                               ; done return

ClearJump:
;           or      P2,#10000000b       ; turn of the flag bit for clear radio
            jp      ClearRadio                 ; clear the radio signal

      .IF   UseSiminor

SimRadio:

            tm      rtimeah, #10000000b ; Test for inactive greater than active
            jr      nz, SimBitZero             ; If so, binary zero received

SimBitOne:

            scf                                ; Set the bit
            jr      RotateInBit                ;

SimBitZero:

            rcf

RotateInBit:

            rrc     CodeT0              ; Shift the new bit into the
            rrc     CodeT1              ; radio word
            rrc     CodeT2              ;
            rrc     CodeT3              ;
            rrc     CodeT4              ;
            rrc     CodeT5              ;

            inc     radioc              ; increase the counter

            cp      radioc, #(49 + 128) ; Test for all 48 bits received
            jp      ugt, CLEARRADIO            ;
            jp      z, KnowSimCode             ;
            jp      RADIO_EXIT                 ;
```

```
CheckSiminor:
             tm     RadioMode, #ROLL_MASK      ; If not in a rolling mode,
             jr     z, INSIG                   ; then it can't be a Siminor transmitter
             cp     RadioTimeOut, #35   ; If the blank time is longer than 35 ms,
             jr     ugt, INSIG                 ; then it can't be a Siminor unit

             or     RadioC, #10000000b  ; Set the flag for a Siminor signal
             clr    ID_B                       ; No ID bits for Siminor
       .ENDIF
INSIG:
             AND    RFlag,#11011111B           ;clear blank time good flag
             cp     rtimeih,#014H       ; test for the max width 5.16
             jr     uge,ClearJump       ; if too wide clear
             cp     rtimeih,#00h        ; test for the min width
             jr     z,ClearJump                ; if high byte is zero, pulse too narrow
ISigOk:
             cp     rtimeah,#014H       ; test for the max width
             jr     uge,ClearJump       ; if too wide clear
             cp     rtimeah,#00h        ; if greater then 0 then signal ok
             jr     z,ClearJump                ; if too narrow clear
ASigOk:
             sub    rtimeal,rtimeil            ; find the difference
             sbc    rtimeah,rtimeih

       .IF   UseSiminor

             tm     RadioC, #10000000b  ; If this is a Siminor code,
             jr     nz, SimRadio        ; then handle it appropriately

       .ENDIF

             tm     rtimeah,#10000000b  ; find out if neg
             jr     nz,NEGDIFF2                ; use 1 for ABC or D
             jr     POSDIFF2
POSDIFF2:
             cp     rtimeah, BitThresh  ; test for 3/2
             jr     ult,BITIS2                 ; mark as a 2
             jr     BITIS3

NEGDIFF2:
             com    rtimeah                    ; invert
             cp     rtimeah, BitThresh  ; test for 2/1
             jr     ult,BIT2COMP        ; mark as a 2
             jr     BITIS1
BITIS3:
             ld     RADIOBIT,#2h        ; set the value
             jr     GOTRADBIT
BIT2COMP:
             com    rtimeah                    ; invert
BITIS2:
             ld     RADIOBIT,#1h        ; set the value
             jr     GOTRADBIT
BITIS1:
             com    rtimeah                    ; invert
             ld     RADIOBIT,#0h        ; set the value
GOTRADBIT:
             clr    rtimeah                    ; clear the time
             clr    rtimeal
             clr    rtimeih
             clr    rtimeil
;            ei                                ; enable interrupts --REDUNDANT
ADDRADBIT:
             SetRpToRadio2Group         ;Macro for assembler error
;            srp    #Radio2Group        ;   -- this is what it does
             tm     rflag,#01000000b           ; test for radio 1 / 3
             jr     nz,RC1INC                  ;

RC3INC:
             tm     RadioMode, #ROLL_MASK      ;If in fixed mode,
            jr      z, Radio3F                ;   no number counter exists
            tm      RadioC,#00000001b         ; test for even odd number
            jr      nz,COUNT3INC          ; if EVEN number counter

Radio3INC:                                    ; else radio

            call    GETTRUEFIX                ;Get the true fixed bit
            cp      RadioC,#14                ; test the radio counter for the specials
            jr      uge,SPECIAL_BITS          ; save the special bits seperate
Radio3R:
Radio3F:
            srp     #RadioGroup
            di                                ; Disable interrupts to avoid pointer collision
            ld      pointerh,#Radio3H         ; get the pointer
            ld      pointerl,#Radio3L         ;
            jr      AddAll
SPECIAL_BITS:
            cp      RadioC,#20                ; test for the switch id
            jr      z,SWITCHID                ; if so then branch

            ld      RTempH,id_b               ; save the special bit
            add     id_b,RTempH               ; *3
            add     id_b,RTempH               ; *3
            add     id_b,radiobit         ; add in the new value
            jr      Radio3R
SWITCHID:
            cp      id_b,#18                  ; If this was a touch code,
            jr      uge, Radio3R          ; then we already have the ID bit
            ld      sw_b,radiobit         ; save the switch ID
            jr      Radio3R

RC1INC:
            tm      RadioMode, #ROLL_MASK     ;If in fixed mode, no number counter
            jr      z, Radio1F
            tm      RadioC,#00000001b         ; test for even odd number
            jr      nz,COUNT1INC          ; if odd number counter

Radio1INC:                                    ; else radio
            call    GETTRUEFIX                ;Get the real fixed code
            cp      RadioC, #02               ;If this is bit 1 of the 1ms code,
            jr      nz, Radio1F               ;then see if we need the switch ID bit
            tm      rflag, #00010000b         ;If this is the first word received,
            jr      z, SwitchBit1         ;then save the switch bit regardless
            cp      id_b, #18                 ;If we have a touch code,
            jr      ult, Radio1F          ;then this is our switch ID bit
SwitchBit1:
            ld      sw_b, radiobit            ;Save touch code ID bit
Radio1F:
            srp     #RadioGroup
            di                                ; Disable interrupts to avoid pointer collision
            ld      pointerh,#Radio1H         ; get the pointer
            ld      pointerl,#Radio1L         ;
            jr      AddAll

GETTRUEFIX:

            ; Chamberlain proprietary fixed code
            ; bit decryption algorithm goes here

            ret

COUNT3INC:
            ld      rollbit, radiobit         ;Store the rolling bit
            srp     #RadioGroup
            di                                ; Disable interrupts to avoid pointer collision
            ld      pointerh,#COUNT3H         ; get the pointer
            ld      pointerl,#COUNT3L         ;
            jr      AddAll
COUNT1INC:
```

```
            ld      rollbit, radiobit           ;Store the rolling bit
            srp     #RadioGroup
            di                                  ; Disable interrupts to avoid pointer collision
            ld      pointerh,#COUNT1H           ; g t the pointers
            ld      pointerl,#COUNT1L           ;
;           jr      AddAll

AddAll:
            ld      addvalueh,@pointerh ; get the value
            ld      addvaluel,@pointerl ;

            add     addvaluel,@pcinterl ; add x2
            adc     addvalueh,@pointerh ;
            add     addvaluel,@pointerl ; add x3
            adc     addvalueh,@pcinterh ;
            add     addvaluel,RADIOBIT  ; add in new number
            adc     addvalueh,#00h              ;
            ld      @pointerh,addvalueh ; save the value
            ld      @pointerl,addvaluel ;
            ei                                  ; Re-enable interrupts
ALLADDED:
            inc     radioc              ; increase the counter
FULLWORD?:
            cp      radioc, MaxBits             ; test for full (10/20 bit) word
            jp      nz,RRETURN                  ; if not then return

            ;;;;;Disable interrupts until word is handled
            or      SKIPRADIO, #NOINT           ; Set the flag to disable radio interrupts
            .IF     TwoThirtyThree
            and     IMR,#11111110B              ; turn off the radio interrupt
            .ELSE
            and     IMR,#11111100B              ; Turn off the radio interrupt
            .ENDIF

            clr     RadioTimeOut        ; Reset the blank time
            cp      RADIOBIT, #00H              ; If the last bit is zero,
            jp      z, ISCCODE                  ;   then the code is the obsolete C code
            and     RFlag,#11111101B            ; Last digit isn't zero, clear B code flag
ISCCODE:
            tm      RFlag,#00010000B            ; test flag for previous word received
            jr      nz,KNOWCODE                 ; if the second word received
FIRST20:
            or      RFlag,#00010000B            ; set the flag
            clr     radioc              ; clear the radio counter
            jp      RRETURN                     ; return

      .IF UseSiminor


 KnowSimCode:

      ; Siminor proprietary rolling code decryption algorithm goes here

      ld     radiolh, #0FFH                     ; Set the code to be incompatible with
      clr    MirrorA                            ; the Chamberlain rolling code
      clr    MirrorB                            ;
      jp     CounterCorrected                   ;

      .ENDIF



 KNOWCODE:

      tm     RadioMode, #ROLL_MASK      ;If not in rolling mode,
      jr     z, CounterCorrected ;    forget the number counter

      ; Chamberlain proprietary counter decryption algorithm goes here
```

```
CounterCorrected:

                srp     #RadioGroup                 ;
                clr     RRTO                        ; clear the got a radio flag
                tm      SKIPRADIO,#NOEECOMM ; test for the skip flag
                jp      nz,CLEARRADIO       ; if skip flag is active then donot look at EE mem

                cp      ID_B, #18                   ;If the ID bits total more than 18,
                jr      ult, NoTCode        ;
                or      RFlag, #00000100b           ;then indicate a touch code
NoTCode:
                ld      ADDRESS,#VACATIONADDR       ; set the non vol address to the VAC flag
                call    READMEMORY                  ; read the value
                ld      VACFLAG,MTEMPH              ; save into volital
                cp      CodeFlag,#REGLEARN  ; test for in learn mode
                jp      nz,TESTCODE                 ; if out of learn mode then test for matching
STORECODE:
                tm      RadioMode, #ROLL_MASK       ;If we are in fixed mode,
                jr      z, FixedOnly        ;then don't compare the counters

CompareCounters:

                cp      PCounterA, MirrorA  ; Test for counter match to previous
                jp      nz, STORENOTMATCH           ; if no match, try again
                cp      PCounterB, MirrorB  ; Test for counter match to previous
                jp      nz, STORENOTMATCH           ; if no match, try again
                cp      PCounterC, MirrorC  ; Test for counter match to previous
                jp      nz, STORENOTMATCH           ; if no match, try again
                cp      PCounterD, MirrorD  ; Test for counter match to previous
                jp      nz, STORENOTMATCH           ; if no match, try again

FixedOnly:
                cp      PRADIO1H,radio1h            ; test for the match
                jp      nz,STORENOTMATCH            ; if not a match then loop again
                cp      PRADIO1L,radio1l            ; test for the match
                jp      nz,STORENOTMATCH            ; if not a match then loop again
                cp      PRADIO3H,radio3h            ; test for the match
                jp      nz,STORENOTMATCH            ; if not a match then loop again
                cp      PRADIO3L,radio3l            ; test for the match
                jp      nz,STORENOTMATCH            ; if not a match then loop again

                cp      AUXLEARNSW, #116            ; If learn was not from wall control,
                jr      ugt, CMDONLY        ; then learn a command only

CmdNotOpen:

                tm      CMD_DEB, #10000000b ; If the command switch is held,
                jr      nz, CmdOrOCS        ;   then we are learning command or o/c/s

CheckLight:

                tm      LIGHT_DEB, #10000000b       ; If the light switch and the lock
                jp      z, CLEARRADIO2              ;     switch are being held,
                tm      VAC_DEB, #10000000b ;     then learn a light trans.
                jp      z, CLEARRADIO2              ;
LearningLight:
                tm      RadioMode, #ROLL_MASK       ; Only learn a light trans. if we are in
                jr      z, CMDONLY                  ;     the rolling mode.
                ld      CodeFlag, #LRNLIGHT ;
                ld      BitMask, #01010101b ;
                jr      CMDONLY

CmdOrOCS:

                tm      LIGHT_DEB, #10000000b       ; If the light switch isn't being held,
                jr      nz, CMDONLY                 ; then see if we are learning o/c/s

CheckOCS:
```

```
                tm      VAC_DEB, #10000000b ; If the vacation switch isn't held,
                jp      z, CLEARRADIO2           ; then it must be a normal command
                tm      RadioMode, #ROLL_MASK    ; Only learn an o/c/s if we are in
                jr      z, CMDONLY               ;    the rolling mode.
                tm      RadioC, #10000000b  ; If the bit for siminor is set,
                jr      nz, CMDONLY              ; then don't learn as an o/c/s Tx
                ld      CodeFlag, #LRNOCS        ; Set flag to learn o/c/s
                ld      BitMask, #10101010b ;

CMDONLY:
                call    TESTCODES                ; test the code to see if in memory now
                cp      ADDRESS, #0FFH           ; If the code isn't in memory
                jr      z, STOREMATCH       ;
WriteOverOCS:
                dec     ADDRESS                  ;
                jp      READYTOWRITE        ;

STOREMATCH:
                cp      RadioMode, #ROLL_TEST    ; If we are not testing a new mode,
                jr      ugt, SameRadioMode  ; then don't switch

                ld      ADDRESS, #MODEADDR  ; Fetch the old radio mode,
                call    READMEMORY               ; change only the low order
                tm      RadioMode, #ROLL_MASK    ; byte, and write in its new value.
                jr      nz, SetAsRoll       ;
SetAsFixed:
                ld      RadioMode, #FIXED_MODE   ;
                call    FixedNums                ; Set the fixed thresholds permanently
                jr      WriteMode                ;
SetAsRoll:
                ld      RadioMode, #ROLL_MODE    ;
                call    RollNums                 ; Set the rolling thresholds permanently
WriteMode:
                ld      MTEMPL, RadioMode        ;
                call    WRITEMEMORY              ;

SameRadioMode:

                tm      RFlag, #00000010B        ; If the flag for the C code is set,
                jp      nz, CCODE                ;    then set the C Code address
                tm      RFlag,#00000100B         ; test for the b code
                jr      nz,BCODE                 ; if a B code jump
ACODE:
                ld      ADDRESS,#2BH        ; set the address to read the last written
                call    READMEMORY               ; read the memory
                inc     MTEMPH              ; add 2 to the last written
                inc     MTEMPH              ;
                tm      RadioMode, #ROLL_MASK    ; If the radio is in fixed mode,
                jr      z, FixedMem              ; then handle the fixed mode memory

RollMem:
                inc     MTEMPH              ; Add another 2 to the last written
                inc     MTEMPH
                and     MTEMPH,#11111100B        ; Set to a multiple of four
                cp      MTEMPH,#1FH              ; test for the last address
                jr      ult, GOTAADDRESS         ; If not the last address jump
                jr      AddressZero              ; Address is now zero

FixedMem:
                and     MTEMPH,#11111110B        ; set the address on a even number
                cp      MTEMPH,#17H              ; test for the last address
                jr      ult,GOTAADDRESS          ; if not the last address jump

AddressZero:
                ld      MTEMPH,#00               ; set the address to 0
GOTAADDRESS:
                ld      ADDRESS,#2BH        ; set the address to write the last written
                ld      RTemp,MTEMPH        ; save the address
                LD      MTEMPL,MTEMPH       ; both bytes same
```

```
        call    WRITEMEMORY             ; write it
        ld      ADDRESS,rtemp       ; set the address
        jr      READYTOWRITE        ;
CCODE:
        tm      RadioMode, #ROLL_MASK       ; If in rolling code mode,
        jp      nz, CLEARRADIO              ; th n HOW DID WE GET A C CODE?
        ld      ADDRESS, #01AH              ; Set the C code address
        jr      READYTOWRITE        ; Store the C code

BCODE:
        tm      RadioMode, #ROLL_MASK       ; If in fixed mode,
        jr      z, BFixed                   ; handle normal touch code
BRoll:
        cp      SW_B, #ENTER        ; If the user is trying to learn a key
        jp      nz, CLEARRADIO              ; other than enter, THROW IT OUT
        ld      ADDRESS, #20H       ; Set the address for the rolling touch code
        jr      READYTOWRITE
BFixed:
        cp      radio3h,#90H        ; test for the 00 code
        jr      nz,BCODEOK                  ;
        cp      radio3l,#29H        ; test for the 00 code
        jr      nz,BCODEOK                  ;
        jp      CLEARRADIO                  ; SKIP MAGIC NUMBER
BCODEOK:
        ld      ADDRESS,#18H        ; set the address for the B code
READYTOWRITE:
        call    WRITECODE                   ; write the code in radio1 and radio3
NOFIXSTORE:
        tm      RadioMode, #ROLL_MASK       ; If we are in fixed mode,
        jr      z, NOWRITESTORE             ; then we are done
        inc     ADDRESS                     ; Point to the counter address
        ld      Radio1H, MirrorA            ; Store the counter into the radio
        ld      Radio1L, MirrorB            ; for the writecode routine
        ld      Radio3H, MirrorC            ;
        ld      Radio3L, MirrorD            ;
        call    WRITECODE

        call    SetMask
        com     BitMask
        ld      ADDRESS, #RTYPEADDR ; Fetch the radio types
        call    READMEMORY                  ;

        tm      RFlag, #10000000b           ; Find the proper byte of the type
        jr      nz, UpByte                  ;
LowByte:
        and     MTEMPL, BitMask             ; Wipe out the proper bits
        jr      MaskDone                    ;
UpByte:
        and     MTEMPH, BitMask             ;
MaskDone:
        com     BitMask                     ;

        cp      CodeFlag, #LRNLIGHT ; If we are learning a light
        jr      z, LearnLight       ; set the appropriate bits
        cp      CodeFlag, #LRNOCS           ; If we are learning an o/c/s,
        jr      z, LearnOCS                 ; set the appropriate bits

Normal:
        clr     BitMask                     ; Set the proper bits as command
        jr      BMReady

LearnLight:
        and     BitMask, #01010101b ; Set the proper bits as worklight
        jr      BMReady                     ; Bit mask is ready
LearnOCS:
        cp      SW_B, #02H                  ; If 'open' switch is not being held,
        jp      nz, CLEARRADIO2             ; then don't accept the transmitter
        and     BitMask,#10101010b  ; Set the proper bits as open/close/stop
```

```
BMReady:
            tm      RFlag, #10000000b           ; Find the proper byte of the type
            jr      nz, UpByt2                  ;
LowByt2:
            or      MTEMPL, BitMask             ; Write the transmitter type in
            jr      MaskDon2                    ;
UpByt2:
            or      MTEMPH, BitMask             ; Write the transmitter type in
MaskDon2:
            call    WRITEMEMORY                 ; Store the transmitter types

NOWRITESTORE:
            xor     p0,#WORKLIGHT       ; toggle light
            or      ledport,#ledh       ; turn off the LED for program mode
            ld      LIGHT1S,#244        ; turn on the 1 second blink
            ld      LEARNT,#0FFH        ; set learnmode timer
            clr     RTO                         ; disallow cmd from learn
            clr     CodeFlag                    ; Clear any learning flags
            jp      CLEARRADIO                  ; return
 STORENOTMATCH:
            ld      PRADIO1H,radio1h            ; transfer radio into past
            ld      PRADIO1L,radio1l            ;
            ld      PRADIO3H,radio3h            ;
            ld      PRADIO3L,radio3l            ;
            tm      RadioMode, #ROLL_MASK       ; If we are in fixed mode,
            jp      z, CLEARRADIO       ; get the next code
            ld      PCounterA, MirrorA  ; transfer counter intc past
            ld      PCounterB, MirrorB  ;
            ld      PCounterC, MirrorC  ;
            ld      PCounterD, MirrorD  ;
            jp      CLEARRADIO
TESTCODE:
            cp      ID_B, #18                   ; If this was a touch code,
            jr      uge, TCReceived             ; handle appropriately

            tm      RFlag, #00000100b           ; If we have received a B code,
            jr      z, AorDCode                 ; then check for the learn mode

            cp      ZZWIN, #64                  ; Test 0000 learn window
            jr      ugt, AorDCode       ; if cut of window nc learn

            cp      Radio1H, #90H       ;
            jr      nz, AorDCode        ;
            cp      Radio1L, #29H       ;
            jr      nz, AorDCode        ;

 ZZLearn:

            push    RP
            srp     #LEARNEE_GRP
            call    SETLEARN
            pop     RP
            jp      CLEARRADIO

 AorDCode:

            cp      L_A_C, #070H        ; Test for in learn limits mode
            jr      uge, FS1                    ; If so, don't blink the LED
            cp      FAULTFLAG,#0FFH             ; test for a active fault
            jr      z,FS1                       ; if a avtive fault skip led set and reset
            and     ledport,#led1       ; turn on the LED for flashing from signal
FS1:
            call    TESTCODES                   ; test the codes
            cp      L_A_C, #070H        ; Test for in learn limits mode
            jr      uge, FS2                    ; If so, don't blink the LED
            cp      FAULTFLAG,#0FFH             ; test for a active fault
            jr      z,FS2                       ; if a avtive fault skip led set and reset
            or      ledport,#ledh       ; turn off the LED for flashing from signal
FS2:
```

```
            cp      ADDRESS,#0FFh       ; test for the not matching state
            jr      nz,GOTMATCH                 ; if matching the send a command if needed
            jp      CLEARRADIO                  ; clear the radio

SimRollCheck:

            inc     ADDRESS                     ; Point to the rolling code
                                                ; (Note:  High word always zero)
            inc     ADDRESS                     ; Point to rest of the counter
            call    READMEMORY                  ; Fetch lower word of counter
            ld      CounterC, MTEMPH            ;
            ld      CounterD, MTEMPL            ;

            cp      CodeT2, CounterC            ; If the two counters are equal,
            jr      nz, UpdateSCode             ; then don't activate
            cp      CodeT3, CounterD            ;
            jr      nz, UpdateSCode             ;
            jp      CLEARRADIO                  ; Counters equal -- throw it out

UpdateSCode:

            ld      MTEMPH, CodeT2              ; Always update the counter if the
            ld      MTEMPL, CodeT3              ; fixed portions match
            call    WRITEMEMORY                 ;

            sub     CodeT3, CounterD            ; Compare the two codes
            sbc     CodeT2, CounterC            ;

            tm      CodeT2, #10000000b  ; If the result is negative,
            jp      nz, CLEARRADIO              ; then don't activate
            jp      MatchGoodSim        ; Match good -- handle normally

GOTMATCH:
            tm      RadioMode, #ROLL_MASK       ; If we are in fixed mode,
            jr      z, MatchGood2       ; then the match is already valid

            tm      RadioC, #10000000b  ; If this was a Siminor transmitter,
            jr      nz, SimRollCheck            ; then test the roll in its own way

            tm      BitMask, #10101010b ; If this was NOT an open/close/stop trans,
            jr      z, RollCheckB       ; then we must check the rolling value

            cp      SW_B, #02                   ; If the o/c/s had a key other than '2'
            jr      nz, MatchGoodOCS            ; then don't check / update the roll

RollCheckB:

            call    TestCounter                 ; Rolling mode -- compare the counter values
            cp      CMP, #EQUAL                 ; If the code is equal,
            jp      z, NOTNEWMATCH              ; then just keep it
            cp      CMP, #FWDWIN        ; If we are not in forward window,
            jp      nz, CheckPast       ; then forget the code
MatchGood:
            ld      Radio1H, MirrorA            ; Store the counter into memory
            ld      Radio1L, MirrorB            ; to keep the roll current
            ld      Radio3H, MirrorC            ;
            ld      Radio3L, MirrorD            ;
            dec     ADDRESS                     ; Line up the address for writing
            call    WRITECODE                   ;

MatchGoodOCS:
MatchGoodSim:

            or      RFlag,#00000001B            ; set the flag for recieving without error
            cp      RTO,#RDROPTIME              ; test for the timer time out
            jp      ult,NOTNEWMATCH             ; if the timer is active then donot reissue cmd

            cp      ADDRESS, #23H       ; If the code was the rolling touch code,
            jr      z, MatchGood2       ; then we already know the transmitter type
            call    SetMask                 ; Set th  mask bits prop rly
            ld      ADDRESS, #RTYPEADDR ; Fetch the transmitter config. bits
            call    READMEMORY              ;
            tm      RFlag, #10000000b       ; If we are in the upper word,
            jr      nz, UpperD              ; check the upper transmitters
LowerD:
            and     BitMask, MTEMPL         ; Isolate our transmitter
            jr      TransType               ; Check out transmitter type
UpperD:
            and     BitMask, MTEMPH         ; Isolate our transmitter
TransType:
            tm      BitMask, #01010101b ; Test for light transmitter
            jr      nz, LightTrans          ; Execute light transmitter
            tm      BitMask, #10101010b ; Test for Open/Close/Stop Transmitter
            jr      nz, OCSTrans        ; Execute open/close/stop transmitter
                                            ; Otherwise, standard command transmitter
MatchGood2:
            or      RFlag,#00000001B        ; set the flag for recieving without error
            cp      RTO,#RDROPTIME          ; test for the timer time out
            jp      ult,NOTNEWMATCH         ; if the timer is active then donot reissue cmd
TESTVAC:
            cp      VACFLAG,#10B        ; test for the vacation mode
            jp      z,TSTSDISABLE       ; if not in vacation mode test the system disable

            tm      RadioMode, #ROLL_MASK   ;
            jr      z, FixedB

            cp      ADDRESS,#23H        ; If this was a touch code,
            jp      nz, NOTNEWMATCH         ; then do a command
            jp      TSTSDISABLE             ;

FixedB:

            cp      ADDRESS,#19H        ; test for the B code
            jp      nz,NOTNEWMATCH          ; if not a B not a match

TSTSDISABLE:
            cp      SDISABLE,#32        ; test for 4 second
            jp      ult,NOTNEWMATCH         ; if 6 s not up not a new code
            clr     RTO                     ; clear the radio timeout
            cp      ONEP2,#00               ; test for the 1.2 second time out
            jp      nz,NOTNEWMATCH          ; if the timer is active then skip the command
RADIOCOMMAND:
            clr     RTO                     ; clear the radio timeout
            tm      RFlag,#00000100b        ; test for a B code
            jr      z,BDONTSET              ; if not a b code donot set flag
zzwinclr:
            clr     ZZWIN                   ; flag got matching B code

            ld      CodeFlag,#BRECEIVED ; flag for aobs bypass
BDONTSET:

            cp      L_A_C, #070H        ; If we were positioning the up limit,
            jr      ult, NormalRadio        ; then start the learn cycle
            jr      z, FirstLearn       ;
            cp      L_A_C, #071H        ; If we had an error,
            jp      nz, CLEARRADIO          ; re-learn, otherwise ignore
R L arning:
            ld      L_A_C, #072H        ; Set the re-learn state
            call    SET_UP_DIR_STATE        ;
            jp      CLEARRADIO              ;
FirstLearn:
            ld      L_A_C, #073H        ; Set the learn state
            call    SET_UP_POS_STATE        ; Start from the "up limit"
            jp      CLEARRADIO              ;

NormalRadio:
            clr     LAST_CMD                ; mark the last command as radio
            ld      RADIO_CMD,#0AAH              ; set the radio command
            jp      CLEARRADIO                   ; return

LightTrans:
            clr     RTO                          ; Clear the radio timeout
            cp      ONEP2,#00                    ; Test for the 1.2 sec. time out
            jp      nz, NOTNEWMATCH              ; If it isn't timed out, leave
            ld      SW_DATA, #LIGHT_SW   ; Set a light command
            jp      CLEARRADIO                   ; return

OCSTrans:
            cp      SDISABLE, #32        ; Test for 4 second system disable
            jp      ult, NOTNEWMATCH             ; if not done not a new code
            cp      VACFLAG, #00H        ; If we are in vacation mode,
            jp      nz, NOTNEWMATCH              ; don't obey the transmitter
            clr     RTO                          ; Clear the radio timeout
            cp      ONEP2, #00                   ; test for the 1.2 second timeout
            jp      nz, NOTNEWMATCH              ; If the timer is active the skip command

            cp      SW_B, #02                    ; If the open button is pressed,
            jr      nz, CloseOrStop              ; then process it

OpenButton:

            cp      STATE, #STOP         ; If we are stopped or
            jr      z, OpenUp                    ; at the down limit, then
            cp      STATE, #DN_POSITION ; begin to move up
            jr      z, OpenUp                    ;
            cp      STATE, #DN_DIRECTION         ; If we are moving down,
            jr      nz, OCSExit                  ; then autoreverse
            ld      REASON, #010H        ; Set the reason as radio
            call    SET_AREV_STATE               ;
            jr      OCSExit                      ;

OpenUp:
            ld      REASON, #010H        ; Set the reason as radio
            call    SET_UP_DIR_STATE             ;
OCSExit:
            jp      CLEARRADIO                   ;

CloseOrStop:

            cp      SW_B, #01                    ; If the stop button is pressed,
            jr      nz, CloseButton              ; then process it

StopButton:

            cp      STATE, #UP_DIRECTION         ; If we are moving or in
            jr      z, StopIt                    ; the autoreverse state,
            cp      STATE, #DN_DIRECTION         ; then stop the door
            jr      z, StopIt                    ;
            cp      STATE, #AUTO_REV             ;
            jr      z, StopIt
            jr      OCSExit

StopIt:

            ld      REASON, #010H        ; Set the reason as radio
            call    SET_STOP_STATE
            jr      OCSExit

CloseButton:

            cp      STATE, #UP_POSITION ; If we are at the up limit
            jr      z, CloseIt                   ; or stopped in travel,
            cp      STATE, #STOP         ; then send the door down
            jr      z, CloseIt                   ;
            jr      OCSExit
```

```
CloseIt:

            ld      REASON, #010H        ; Set the reason as radio
            call    SET_DN_DIR_STATE
            jr      OCSExit

SetMask:

            and     RFlag, #01111111b          ; Reset the page 1 bit
            tm      ADDRESS, #11110000b ; If our address is on page 1,
            jr      z, InLowerByte             ; then set the proper flag
            or      RFlag, #10000000b          ;
InLowerByte:
            tm      ADDRESS, #00001000b ; Binary search to set the
            jr      z, ZeroOrFour       ; proper bits in the bit mask
EightOrTwelve:
            ld      BitMask, #11110000b
            jr      LSNybble
ZeroOrFour:
            ld      BitMask, #00001111b ;
LSNybble:
            tm      ADDRESS, #00000100b
            jr      z, ZeroOrEight
FourOrTwelve:
            and     BitMask, #11111100b ;
            ret
ZeroOrEight:
            and     BitMask, #00110011b ;
            ret

TESTCODES:
            ld      ADDRESS, #RTYPEADDR ; Get the radio types
            call    READMEMORY                 ;
            ld      RadioTypes, MTEMPL  ;
            ld      RTypes2, MTEMPH            ;
            tm      RadioMode, #ROLL_MASK      ;
            jr      nz, RollCheck       ;
            clr     RadioTypes                 ;
            clr     RTypes2
RollCheck:
            clr     ADDRESS                    ; start address is 0
NEXTCODE:
            call    SetMask                    ; Get the approprite bit mask
            and     BitMask, RadioTypes ; Isolate the current transmitter types
HAVEMASK:
            call    READMEMORY                 ; read the word at this address
            cp      MTEMPH,radio1h             ; test for the match
            jr      nz,NOMATCH                 ; if not matching then do next address
            cp      MTEMPL,radio1l             ; test for the match
            jr      nz,NOMATCH                 ; if not matching then do next address
            inc     ADDRESS                    ; set the second half of the code
            call    READMEMORY                 ; read the word at this address
            tm      BitMask, #10101010b ; If this is an Open/Close/Stop trans.,
            jr      nz, CheckOCS1       ; then do the different check
            cp      CodeFlag, #LRNOCS          ; If we are in open/close/stop learn mode,
            jr      z, CheckOCS1        ; then do the different check
            cp      MTEMPH,radio3h             ; test for the match
            jr      nz,NOMATCH2                ; if not matching then do the next address
            cp      MTEMPL,radio3l             ; test for the match
            jr      nz,NOMATCH2                ; if not matching then do the next address

            ret                                ; return with the address of the match

CheckOCS1:

            sub     MTEMPL, radio3l            ; Subtract the radio from the memory
            sbc     MTEMPH, radio3h            ;
            cp      CodeFlag, #LRNOCS          ; If we are trying to learn open/close/stop,
            jr      nz, Positive        ; then we must complement to be positive
            com     MTEMPL              ;
            com     MTEMPH              ;
            add     MTEMPL, #1                  ; Switch from ones complement to 2's
            adc     MTEMPH, #0                  ; complement
Positive:
            cp      MTEMPH, #00                 ; We must be within 2 to match properly
            jr      nz, NOMATCH2        ;
            cp      MTEMPL, #02                 ;
            jr      ugt, NOMATCH2       ;

            ret                                 ; Return with the address of the match

NOMATCH:
            inc     ADDRESS                     ; set the address to the next code
NOMATCH2:
            inc     ADDRESS                     ; set the address to the next code
            tm      RadioMode, #ROLL_MASK       ; If we are in fixed mode,
            jr      z, AtNextAdd        ; then we are at the next address
            inc     ADDRESS                     ; Roll mode -- advance past the counter
            inc     ADDRESS                     ;
            cp      ADDRESS,#10H        ; If we are on the second page
            jr      nz, AtNextAdd       ; then get the other tx. types
            ld      RadioTypes, RTypes2 ;
AtNextAdd:
            cp      ADDRESS,#22H        ; test for the last address
            jr      ult,NEXTCODE        ; if not the last address then try again

GOTNOMATCH:
            ld      ADDRESS,#0FFH       ; set the no match flag
            ret                                 ; and return

NOTNEWMATCH:
            clr     RTC                         ; reset the radio time out
            and     RFlag,#00000001B            ; clear radio flags leaving recieving w/o error
            clr     radioc              ; clear the radio bit counter
;           ld      LEARNT,#0FFH        ; set the learn timer "turn off" and backup
            jp      RADIO_EXIT                  ; return

CheckPast:
        ; Proprietary algorithm for maintaining
        ; rolling code counter
        ; Jumps to either MatchGood, UpdatePast or CLEARRADIO

UpdatePast:

            ld      LastMatch, ADDRESS' ; Store the last fixed code received
            ld      PCounterA, MirrorA  ; Store the last counter received
            ld      PCounterB, MirrorB  ;
            ld      PCounterC, MirrorC  ;
            ld      PCounterD, MirrorD  ;
CLEARRADIO2:
            ld      LEARNT, #0FFH       ; Turn off the learn mode timer
            clr     CodeFlag

CLEARRADIO:

            .IF     TwoThirtyThree
            and     IRQ,#00111111B              ; clear the bit setting direction to neg edge
            .ENDIF

            ld      RINFILTER,#0FFH             ; set flag to active
CLEARRADIOA:
            tm      RFlag,#00000001B            ; test for receiving without error
            jr      z,SKIPRTC                   ; if flag not set then donot clear timer
            clr     RTC                         ; clear radio timer
SKIPRTC:
            clr     radioc              ; clear the radio counter
            clr     RFlag                       ; clear the radio flag
```

```
;               clr     ID_B                    . Clear the ID bits
                jp      RADIO_EXIT              ; return

TCReceived:

                cp      L_A_C, #070H        ; Test for in learn limits mode
                jr      uge, TestTruncate       ; If so, don't blink the LED
                cp      FAULTFLAG, #0FFH        ; If no fault
                jr      z, TestTruncate         ; turn on the led
                and     ledport, #ledl          ;
                jr      TestTruncate        ; Truncate off most significant digit

TruncTC:

                sub     Radio1L, #0E3h          ; Subtract out 3^9 to truncate
                sbc     Radio1H, #04Ch          ;

TestTruncate:

                cp      Radio1H, #04Ch          ; If we are greater than 3^9,
                jr      ugt, TruncTC        ; truncate down
                jr      ult, GotTC              ;
                cp      Radio1L, #0E3h          ;
                jr      uge, TruncTC        ;

GotTC:

                ld      ADDRESS, #TOUCHID       ; Check to make sure the ID code is good
                call    READMEMORY              ;
                cp      L_A_C, #070H        ; Test for in learn limits mode
                jr      uge, CheckID        ; If so, don't blink the LED
                cp      FAULTFLAG, #0FFH        ; If no fault,
                jr      z, CheckID              ; turn off the LED
                or      ledport, #ledh          ;
CheckID:
                cp      MTEMPH, Radio3H         ;
                jr      nz, CLEARRADIO          ;
                cp      MTEMPL, Radio3L         ;
                jr      nz, CLEARRADIO          ;

                call    TestCounter             ; Test the rolling code counter
                cp      CMP, #EQUAL             ; If the counter is equal,
                jp      z, NOTNEWMATCH          ; then call it the same code
                cp      CMP, #FWDWIN        ;
                jr      nz, CLEARRADIO          ;

                ; Counter good -- update it

                ld      COUNT1H, Radio1H        ; Back up radio code
                ld      COUNT1L, Radio1L        ;

                ld      Radio1H, MirrorA        ;Write the counter
                ld      Radio1L, MirrorB        ;
                ld      Radio3H, MirrorC        ;
                ld      Radio3L, MirrorD        ;
                dec     ADDRESS
                call    WRITECODE               ;

                ld      Radio1H, COUNT1H        ; Restore the radio code
                ld      Radio1L, COUNT1L        ;

                cp      CodeFlag, #NORMAL       ; Find and jump to current mode
                jr      z, NormTC               ;
                cp      CodeFlag, #LRNTEMP  ;
                jp      z, LearnTMP             ;
                cp      CodeFlag, #LRNDURTN ;
                jp      z, LearnDur             ;
                jp      CLEARRADIO              ;
```

```
NormTC:

            ld     ADDRESS, #TOUCHPERM ; Compare the four-digit touch
            call   READMEMORY                 ; code to our permanent password
            cp     Radio1H, MTEMPH            ;
            jr     nz, CheckTCTemp            ;
            cp     Radio1L, MTEMPL            ;
            jr     nz, CheckTCTemp            ;

            cp     SW_B, #ENTER        ; If the ENTER key was pressed,
            jp     z, RADIOCOMMAND            ; issue a B code radio command
            cp     SW_B, #POUND        ; If the user pressed the pound key,
            jr     z, TCLearn                 ; enter the learn mode
            ; Star key pressed -- start 30 s timer

            clr    LEARNT              ;
            ld     FLASH_COUNTER, #06h ; Blink the worklight three
            ld     FLASH_DELAY,#FLASH_TIME    ; times quickly
            ld     FLASH_FLAG, #0FFH          ;
            ld     CodeFlag, #LRNTEMP  ; Enter learn temporary mode
            jp     CLEARRADIO                 ;

TCLearn:

            ld     FLASH_COUNTER, #04h ; Blink the worklight two
            ld     FLASH_DELAY,#FLASH_TIME    ; times quickly
            ld     FLASH_FLAG, #0FFH          ;

            push   RP                         ; Enter learn mode
            srp    #LEARNEE_GRP
            call   SETLEARN
            pop    RP

            jp     CLEARRADIO

CheckTCTemp:

            ld     ADDRESS, #TOUCHTEMP ; Compare the four-digit touch
            call   READMEMORY                 ; code to our temporary password
            cp     Radio1H, MTEMPH            ;
            jp     nz, CLEARRADIO             ;
            cp     Radio1L, MTEMPL            ;
            jp     nz, CLEARRADIO             ;

            cp     STATE, #DN_POSITION ; If we are not at the down limit,
            jp     nz, RADIOCOMMAND           ; issue a command regardless

            ld     ADDRESS, #DURAT            ; If the duration is at zero,
            call   READMEMORY                 ; then don't issue a command
            cp     MTEMPL, #00                ;
            jp     z, CLEARRADIO       ;

            cp     MTEMPH, #ACTIVATIONS       ; If we are in number of activations
            jp     nz, RADIOCOMMAND           ; mode, then decrement the
            dec    MTEMPL              ; number of activations left
            call   WRITEMEMORY                ;
            jp     RADIOCOMMAND

LearnTMP:
            cp     SW_B, #ENTER        ; If the user pressed a key other
            jp     nz, CLEARRADIO             ; then enter, reject the code

            ld     ADDRESS, #TOUCHPERM ; If the code entered matches the
            call   READMEMORY                 ; permanent touch code,
            cp     Radio1H, MTEMPH            ; then reject the code as a
            jp     nz, TempGood        ; temporary code
            cp     Radio1L, MTEMPL            ;
            jp     z, CLEARRADIO       ;
```

```
TempGood:

             ld     ADDRESS, #TOUCHTEMP ; Write the code into temp.
             ld     MTEMPH, Radio1H            ; code memory
             ld     MTEMPL, Radio1L            ;
             call   WRITEMEMORY                ;

             ld     FLASH_COUNTER, #08h ; Blink the worklight four
             ld     FLASH_DELAY,#FLASH_TIME    ; times quickly
             ld     FLASH_FLAG, #0FFH          ;

             ; Start 30 s timer

             clr    LEARNT
             ld     CodeFlag, #LRNDURTN ; Enter learn duration mode
             jp     CLEARRADIO                 ;

LearnDur:

             cp     Radio1H, #00        ; If the duration was > 255,
             jp     nz, CLEARRADIO             ; reject the duration entered

             cp     SW_B, #POUND        ; If the user pressed the pound
             jr     z, NumDuration             ; key, number of activations mode
             cp     SW_B, #STAR                ; If the star key was pressed,
             jr     z, HoursDur                ; enter the timer mode
             jp     CLEARRADIO                 ; Enter pressed -- reject code

NumDuration:

             ld     MTEMPH, #ACTIVATIONS       ; Flag number of activations mode
             jr     DurationIn                 ;

HoursDur:

             ld     MTEMPH, #HOURS             ; Flag number of hours mode

DurationIn:

             ld     MTEMPL, Radio1L            ; Load in duration
             ld     ADDRESS, #DURAT            ; Write duration and mode
             call   WRITEMEMORY                ; into nonvolatile memory

             ; Give worklight one long blink
             xor    P0, #WORKLIGHT             ; Give the light one blink
             ld     LIGHT1S, #244       ; lasting one second
             clr    CodeFlag                   ; Clear the learn flag
             jp     CLEARRADIO

;----------------------------------------------------------------------
;     Test Rolling Code Counter Subroutine
;     Note:  CounterA-D will be used as temp registers
;
;----------------------------------------------------------------------

TestCounter:
             push   RP
             srp    #CounterGroup
             inc    ADDRESS                    ; Point to the rolling code counter
             call   READMEMORY                 ; Fetch lower word of counter
             ld     countera, MTEMPH
             ld     counterb, MTEMPL
             inc    ADDRESS                    ; Point to rest of the counter
             call   READMEMORY                 ; Fetch upper word of counter
             ld     counterc, MTEMPH
             ld     counterd, MTEMPL

             ;-------------------------------------------------------------
             ;      Subtract old counter (countera-d) from current
```

```
        ;       counter (mirrora-d) and store in countera-d
        ;-------------------------------------------------------------

        com     countera                    ; Obtain twos complement of counter
        com     counterb
        com     counterc
        com     counterd
        add     counterd, #01H
        adc     counterc, #00H
        adc     counterb, #00H
        adc     countera, #00H

        add     counterd, mirrord           ; Subtract
        adc     counterc, mirrorc
        adc     counterb, mirrorb
        adc     countera, mirrora

        ;-------------------------------------------------------------
        ;       If the msb of counterd is negative, check to see
        ;       if we are inside the negative window
        ;-------------------------------------------------------------

        tm      countera, #10000000B
        jr      z, CheckFwdWin

CheckBackWin:

        cp      countera, #0FFH             ; Check to see if we are
        jr      nz, OutOfWindow             ; less than -0400H
        cp      counterb, #0FFH             ; (i.e. are we greater than
        jr      nz, OutOfWindow             ; 0xFFFFFC00H)
        cp      counterc, #0FCH             ;
        jr      ult, OutOfWindow            ;

InBackWin:

        ld      CMP, #BACKWIN       ; Return in back window
        jr      CompDone

CheckFwdWin:

        cp      countera, #00H              ; Check to see if we are less
        jr      nz, OutOfWindow             ; than 0C00  3072 = 1024
        cp      counterb, #00H              ; activations
        jr      nz, OutOfWindow             ;
        cp      counterc, #0CH              ;
        jr      uge, OutOfWindow

        cp      counterc, #00H
        jr      nz, InFwdWin
        cp      counterd, #00H
        jr      nz, InFwdWin

CountersEqual:

        ld      CMP, #EQUAL                 ;Return equal counters
        jr      CompDone

InFwdWin:

        ld      CMP, #FWDWIN        ;Return in forward window
        jr      CompDone

OutOfWindow:

        ld      CMP, #OUTOFWIN              ;Return out of any window

CompDone:

. . . . . .
```

```
              pop     RP
              ret

;***************************************************************
; Clear interrupt
;***************************************************************
ClearRadio:

       cp     RadioMode, #ROLL_TEST            ;If in fixed or rolling mode,
       jr     ugt, MODEDONE               ;    then we cannot switch

       tm     T125MS, #00000001b          ;If our 'coin toss' was a zero,
       jr     z, SETROLL                       ;    set as the rolling mode

SETFIXED:

       ld     RadioMode, #FIXED_TEST
       call   FixedNums
       jp     MODEDONE

SETROLL:

       ld     RadioMode, #ROLL_TEST
       call   RollNums

MODEDONE:

       clr    RadioTimeOut                     ; clear radio timer
       clr    RadioC                           ; clear the radio counter
       clr    RFlag                                   ; clear the radio flags

RETURN:
       pop    RP                               ; reset the RP
       iret                                    ; return

FixedNums:

       ld     BitThresh, #FIXTHR
       ld     SyncThresh, #FIXSYNC
       ld     MaxBits, #FIXBITS
       ret

RollNums:

       ld     BitThresh, #DTHR
       ld     SyncThresh, #DSYNC
       ld     MaxBits, #DBITS
       ret

;***************************************************************************
***********
;  rotate mirror LoopCount  * 2  then add
;***************************************************************************
***********
RotateMirrorAdd:

       rcf                                     ; clear the carry
       rlc    mirrord                                 ;
       rlc    mirrorc                                 ;
       rlc    mirrorb                                 ;
       rlc    mirrora                                 ;
       djnz   loopcount,RotateMirrorAdd        ; loop till done

;***************************************************************************
***********
;  Add mirror to counter
;***************************************************************************
***********
AddMirrorToCounter:
        add     counterd,mirrord                ;
        adc     counterc,mirrorc                ;
        adc     counterb,mirrorb                ;
        adc     countera,mirrora                ;
        ret


;*********************************************************************
; LEARN DEBOUNCES THE LEARN SWITCH 80mS
; TIMES OUT THE LEARN MODE 30 SECONDS
; DEBOUNCES THE LEARN SWITCH FOR ERASE 6 SECONDS
;*********************************************************************
LEARN:
        srp     #LEARNEE_GRP            ; set the register pointer
        cp      STATE,#DN_POSITION      ; test for motor stoped
        jr      z,TESTLEARN                     ;
        cp      STATE,#UP_POSITION              ; test for motor stoped
        jr      z,TESTLEARN                     ;
        cp      STATE,#STOP                     ; test for motor stoped
        jr      z,TESTLEARN                     ;
        cp      L_A_C,#074H                     ; Test for traveling
        jr      z,TESTLEARN                     ;
;       ld      learnt,#0FFH                    ; set the learn timer
        cp      learnt,#241                     ; test for the learn 30 second timeout
        jr      nz,ERASETEST            ; if not then test erase
        jr      learnoff                                ; if 30 seconds then turn off the learn mode
TESTLEARN:
        cp      learndb,#236                    ; test for the debounced release
        jr      nz,LEARNNOTRELEASED             ; if debouncer not released then jump
LEARNRELEASED:
SmartRelease:
        cp      L_A_C, #070H            ; Test for in learn limits mode
        jr      nz, NormLearnBreak      ; If not, treat the break as normal

        ld      REASON, #00H                    ; Set the reason as command
        call    SET_STOP_STATE                  ;

NormLearnBreak:

        clr     LEARNDB                                 ; clear the debouncer

        ret                                     ; return

LEARNNOTRELEASED:
        cp      CodeFlag,#LRNTEMP               ;test for learn mode
        jr      uge,INLEARN                     ; if in learn jump
        cp      learndb,#20                     ; test for debounce period
        jr      nz,ERASETEST            ; if not then test the erase period
SETLEARN:
        call    SmartSet                        ;
ERASETEST:
        cp      L_A_C, #070H            ; Test for in learn limits mode
        jr      uge,ERASERELEASE                ; If so, DON'T ERASE THE MEMORY
        cp      learndb,#0FFH                           ; test for learn button active
        jr      nz,ERASERELEASE                 ; if button released set the erase timer
        cp      eraset,#0FFH                    ; test for timer active
        jr      nz,ERASETIMING                  ; if the timer active jump
        clr     eraset                          ; clear the erase timer
ERASETIMING:
        cp      eraset,#48                      ; test for the erase period
        jr      z,ERASETIME                     ; if timed out the erase
        ret                                     ; else we return
ERASETIME:
        or      ledport,#ledh                           ; turn off the led
        ld      skipradio,#NOEECOMM                     ; set the flag to skip the radio read
        call    CLEARCODES                      ; clear all codes in memory
        clr     skipradio                       ; reset the flag to skip radio

        ld      learnt,#0FFH                    ; set the learn timer
        clr     CodeFlag
        ret                                     ; return

SmartSet:
        cp      L_A_C, #070H                    ; Test for in learn limits mode
        jr      nz, NormLearnMake1              ; If not, treat normally
        ld      REASON, #00H                    ; Set the reason as command
        call    SET_DN_NOBLINK                          ;
        jr      LearnMakeDone                   ;
NormLearnMake1:
        cp      L_A_C, #074H                    ; Test for traveling down
        jr      nz, NormLearnMake2              ; If not, treat normally
        ld      L_A_C, #075H                    ; Reverse off false floor
        ld      REASON, #00H                    ; Set the reason as command
        call    SET_AREV_STATE                          ;
        jr      LearnMakeDone                   ;
NormLearnMake2:
        clr     LEARNT                          ; clear the learn timer
        ld      CodeFlag, #REGLEARN             ; Set the learn flag
        and     ledport,#ledl                           ; turn on the led
        clr     VACFLAG                         ; clear vacation mode
        ld      ADDRESS,#VACATIONADDR                   ; set the non vol address for vacation
        clr     MTEMPH                          ; clear the data for cleared vacation
        clr     MTEMPL                          ;
        ld      SKIPRADIO,#NOEECOMM                     ; set the flag
        call    WRITEMEMORY                ; write the memory
        clr     SKIPRADIO                       ; clear the flag
LearnMakeDone:
        ld      LEARNDB,#0FFH                           ; set the debouncer
        ret

ERASERELEASE:
        ld      eraset,#0FFH                    ; turn off the erase timer
        cp      learndb,#236                    ; test for the debounced release
        jr      z,LEARNRELEASED            ; if debouncer not released then jump
        ret                                     ; return

INLEARN:
        cp      learndb,#20                     ; test for the debounce period
        jr      nz,TESTLEARNTIMER               ; if not then test the learn timer for.time out
        ld      learndb,#0FFH                           ; set the learn db
TESTLEARNTIMER:
        cp      learnt,#240                     ; test for the learn 30 second timeout
        jr      nz,ERASETEST               ; if not then test erase
learnoff:
        or      ledport,#ledh                           ; turn off the led
        ld      learnt,#0FFH                    ; set the learn timer
        ld      learndb,#0FFH                           ; set the learn debounce
        clr     CodeFlag                        ; Clear ANY code types
        jr      ERASETEST                       ; test the erase timer



;*****************************************************************
; WRITE WORD TO MEMORY
; ADDRESS IS SET IN REG ADDRESS
; DATA IS IN REG MTEMPH AND MTEMPL
; RETURN ADDRESS IS UNCHANGED
;*****************************************************************
WRITEMEMORY:
        push    RP                              ; SAVE THE RP
        srp     #LEARNEE_GPP               ; set the register pointer

        call    STARTB                          ; output the start bit
        ld      serial,#00110000B               ; set byte to enable write
        call    SERIALOUT                       ; output the byte
        and     csport,#csl                ; reset the chip select
        call    STARTB                          ; output the start bit
        ld      serial,#01000000B          ; set the byte for write
```

```
       or     serial,address                    ; or in the address
       call   SERIALOUT                         ; output the byte
       ld     serial,mtemph                     ; set the first byte to write
       call   SERIALOUT                         ; output the byte
       ld     serial,mtempl                     ; set the s cond byte to write
       call   SERIALOUT                         ; output th  byt
       call   ENDWRITE                          ; wait for the ready status
       call   STARTB                            ; output the start bit
       ld     serial,#00000000B         ; set byte to disable write
       call   SERIALOUT                         ; output the byte
       and    csport,#csl               ; reset the chip select
       or     P2M_SHADOW,#clockh        ; Change program switch back to read
       ld     P2M,P2M_SHADOW                    ;
       pop    RP                                ; reset the RP
       ret

;*******************************************************************
; READ WORD FROM MEMORY
; ADDRESS IS SET IN REG ADDRESS
; DATA IS RETURNED IN REG MTEMPH AND MTEMPL
; ADDRESS IS UNCHANGED
;*******************************************************************
READMEMORY:
       push   RP                                ;
       srp    #LEARNEE_GRP              ; set the register pointer

       call   STARTB                            ; output the start bit
       ld     serial,#10000000B         ; preamble for read
       or     serial,address                    ; or in the address
       call   SERIALOUT                         ; output the byte
       call   SERIALIN                          ; read the first byte
       ld     mtemph,serial                     ; save the value in mtemph
       call   SERIALIN                          ; read teh second byte
       ld     mtempl,serial                     ; save the value in mtempl
       and    csport,#csl               ; reset the chip select
       or     P2M_SHADOW,#clockh                ; Change program switch back to read
       ld     P2M, P2M_SHADOW                   ;
       pop    RP                                ;
       ret
;*******************************************************************
;WRITE CODE TO 2 MEMORY ADDRESS
; CODE IS IN RADIO1H RADIO1L RADIO3H RADIO3L
;*******************************************************************
WRITECODE:
              push   RP                 ;
              srp    #LEARNEE_GRP ; set the register pointer
              ld     mtemph,Radio1H     ; transfer the data from radio 1 to the temps
              ld     mtempl,Radio1L     ;
              call   WRITEMEMORY        ; write the temp bits
              inc    address            ; next address
              ld     mtemph,Radio3H     ; transfer the data from radio 3 to the temps
              ld     mtempl,Radio3L     ;
              call   WRITEMEMORY        ; write the temps
              pop    RP                 ;
              ret                       ; return


;*******************************************************************
; CLEAR ALL RADIO CODES IN THE MEMORY
;*******************************************************************


CLEARCODES:
       push   RP                                ;
       srp    #LEARNEE_GRP              ; set the register pointer
       ld     MTEMPH,#0FFH              ; set the codes to illegal codes
       ld     MTEMPL,#0FFH              ;
       ld     address,#00H                      ; clear address 0
```

```
CLEARC:
      call   WRITEMEMORY                  ; "A0"
      inc    address                                ; set the next address
      cp     address,#(AddressCounter - 1)          ; test for the last address of radio
      jr     ult,CLEARC
      clr    mtemph                       ; clear data
      clr    mtempl
      call   WRITEMEMORY                            ; Clear radio types
      ld     address,#AddressAPointer     ; clear address F
      call   WRITEMEMORY                  ;

      ld     address,#MODEADDR            ;Set EEPROM memory as fixed test
      call   WRITEMEMORY                  ;

      ld     RadioMode, #FIXED_TEST       ;Revert to fixed mode testing
      ld     BitThresh, #FIXTHR
      ld     SyncThresh, #FIXSYNC
      ld     MaxBits, #FIXBITS

CodesCleared:

      pop    RP                           ;
      ret                                 ; return

;**********************************************************
; START BIT FOR SERIAL NONVOL
; ALSO SETS DATA DIRECTION AND AND CS
;**********************************************************
STARTB:
      and    P2M_SHADOW, #(clockl & dol)            ; Set output mode for clock line and
      ld     P2M,P2M_SHADOW                         ; I/O lines
      and    csport,#csl                  ;
      and    clkport,#clockl                        ; start by clearing the bits
      and    dioport,#dol                 ;
      or     csport,#csh                  ; set the chip select
      or     dioport,#doh                 ; set the data out high
      or     clkport,#clockh                        ; set the clock
      and    clkport,#clockl                        ; reset the clock low
      and    dioport,#dol                 ; set the data low
      ret                                           ; return

;**********************************************************
; END OF CODE WRITE
;**********************************************************
ENDWRITE:
      and    csport,#csl                  ; reset the chip select
      nop                                           ; delay
      or     csport,#csh                  ; set the chip select
      or     P2M_SHADOW, #doh                       ; Set the data line to input
      ld     P2M,P2M_SHADOW                         ; set port 2 mode forcing input mode data
ENDWRITELOOP:
      ld     temph,dioport                          ; read the port
      and    temph,#doh                             ; mask
      jr     z,ENDWRITELOOP                         ; if the bit is low then loop until done
      and    csport,#csl                  ; reset the chip select
      or     P2M_SHADOW, #clockh          ; Reset the clock line to read smart button
      and    P2M_SHADOW, #dol                       ; Set the data line back to output
      ld     P2M,P2M_SHADOW                         ; set port 2 mode forcing output mode
      ret


;**********************************************************
; SERIAL OUT
; OUTPUT THE BYTE IN SERIAL
;**********************************************************
SERIALOUT:
      and    P2M_SHADOW,#(dol & clockl)   ; Set the clock and data lines to outputs
      ld     P2M,P2M_SHADOW                         ; set port 2 mode forcing output mode data
      ld     templ,#8H                              ; set the count for eight bits
```

```
SERIALOUTLOOP:
       rlc    serial                           ; get the bit to output into the carry
       jr     nc,ZEROOUT                              ; output a zero if no carry
ONEOUT:
       or     dioport,#doh                     ; set the data out high
       or     clkport,#clockh                         ; set the clock high
       and    clkport,#clockl                         ; reset the clock low
       and    dioport,#dol                     ; reset the data out low
       djnz   templ,SERIALOUTLOOP
                                                      ; loop till done
       ret                                            ; return
ZEROOUT:
       and    dioport,#dol                     ; reset the data out low
       or     clkport,#clockh                         ; set the clock high
       and    clkport,#clockl                         ; reset the clock low
       and    dioport,#dol                     ; reset the data out low
       djnz   templ,SERIALOUTLOOP
                                                      ; loop till done
       ret                                            ; return


;*************************************************************
; SERIAL IN
; INPUTS A BYTE TO SERIAL
;*************************************************************
SERIALIN:
       or     P2M_SHADOW, #doh                        ; Force the data line to input
       ld     P2M,P2M_SHADOW                          ; set port 2 mode forcing input mode data
       ld     templ,#8H                               ; set the count for eight bits
SERIALINLOOP:
       or     clkport,#clockh                         ; set the clock high
       rcf                                            ; reset the carry flag
       ld     temph,dioport                           ; read the port
       and    temph,#doh                              ; mask out the bits
       jr     z,DONTSET
       scf                                            ; set the carry flag
DONTSET:
       rlc    serial                           ; get the bit into the byte
       and    clkport,#clockl                         ; reset the clock low
       djnz   templ,SERIALINLOOP
                                                      ; loop till done
       ret                                            ; return

;*************************************************************
; TIMER UPDATE FROM INTERUPT EVERY 0.256mS
;*************************************************************
SkipPulse:
;      tm     SKIPRADIO, #NOINT                ;If the 'no radio interrupt'
;      jr     nz, NoPulse                      ;flag is set, just leave
;      or     IMR,#RadioImr                    ; turn on the radio
;NoPulse:
       iret




TIMERUD:

       tm     SKIPRADIO, #NOINT                ;If the 'no radio interrupt'
       jr     nz, NoEnable              ;flag is set, just leave
       or     IMR,#RadioImr             ; turn on the radio
NoEnable:
       decw   T0EXTWORD                        ; decrement the T0 extension

T0ExtDone:

       tm     P2, #LINEINPIN                   ; Test the AC line in
       jr     z, LowAC                         ; If it's low, mark zero crossing
HighAC:
        inc     LineCtr                         ; Count the high time
        jr      LineDone                        ;
LowAC:
        cp      LineCtr, #08                ; If the line was low before
        jr      ult, HighAC                     ; then one-shot the edge of the line
        ld      LinePer, LineCtr                ; Store the high time
        clr     LineCtr                         ; Reset the counter
        ld      PhaseTMR, PhaseTime         ; Reset the timer for the phase control

LineDone:

        cp      PowerLevel, #20                 ; Test for at full wave of phase
        jr      uge, PhaseOn                ; If not, turn off at the start of the phase
        cp      PowerLevel, #00                 ; If we're at the minimum,
        jr      z, PhaseOff                     ; then never turn the phase control on
        dec     PhaseTMR                        ; Update the timer for phase control
        jr      mi, PhaseOn                     ; If we are past the zero point, turn on the line

PhaseOff:
        and     PhasePrt, #~PhaseHigh           ; Turn off the phase control
        jr      PhaseDone                       ;

PhaseOn:
        or      PhasePrt, #PhaseHigh            ; Turn on the phase control

PhaseDone:

        tm      P3, #00000010b                  ; Test the RPM in pin
        jr      nz, IncRPMDE                ; If we're high, increment the filter
DecRPMDE:
        cp      RPM_FILTER, #00                 ; Decrement the value of the filter if
        jr      z, RPMFiltered                  ; we're not already at zero
        dec     RPM_FILTER                      ;
        jr      RPMFiltered                     ;
IncRPMDE:
        inc     RPM_FILTER                      ; Increment the value of the filter
        jr      nz, RPMFiltered                 ; and back turn if necessary
        dec     RPM_FILTER                      ;

RPMFiltered:
        cp      RPM_FILTER, #12                 ; If we've seen 2.5 ms of high time
        jr      z, VectorRPMHigh                ; then vector high
        cp      RPM_FILTER, # 255 - 12          ; If we've seen 2.5 ms of low time
        jr      nz, TaskSwitcher                ; then vector low
VectorRPMLow:
        clr     RPM_FILTER                      ;
        jr      TaskSwitcher                ;
VectorRPMHigh:
        ld      RPM_FILTER, #0FFH               ;

TaskSwitcher

        tm      T0EXT,#00000001b                ; skip everyother pulse
        jr      nz,SkipPulse
        tm      T0EXT,#00000010b                ; Test for odd numbered task
        jr      nz,TASK1357                     ; If so do the 1ms timer update
        tm      T0EXT,#00000100b                ; Test for task 2 or 6
        jr      z, TASK04                       ; If not, then go to Tasks 0 and 4
        tm      T0EXT,#00001000b                ; Test for task 6
        jr      nz, TASK6                       ; If so, jump
                                                ; Otherwise, we must be in task 2

TASK2:
                or      IMR,#RETURN_IMR         ; turn on the interrupt
                ei
                call    STATEMACHINE        ; do the motor function
                iret

TASK04:
```

```
                or      IMR,#RETURN_IMR             ; turn on the interrupt
                ei
                push    rp                          ; save the rp
                srp     #TIMER_GROUP          ; set the rp for the switches
                call    switches                    ; test the switches
                pop     rp
                iret


TASK6:
                or      IMR,#RETURN_IMR             ; turn on the interrupt
                ei
                call    TIMER4MS                    ; do the four ms timer
                iret


TASK1357:
                push    RP
                or      IMR,#RETURN_IMR             ; turn on the interrupt
                ei

ONEMS:

                tm      p0,#DOWN_COMP               ; Test down force pot.
                jr      nz,HigherDn                 ; Average too low -- output pulse
LowerDn:
                and     p3,#(~DOWN_OUT)             ; take pulse output low
                jr      DnPotDone                   ;
HigherDn:
                or      p3,#DOWN_OUT          ; Output a high pulse
                inc     DN_TEMP                     ; Increase measured duty cycle
DnPotDone:
                tm      p0,#UP_COMP                 ; Test the up force pot.
                jr      nz,HigherUp                 ; Average too low -- output pulse
LowerUp:
                and     P3,#(~UP_OUT)         ; Take pulse output low
                jr      UpPotDone                   ;
HigherUp:
                or      P3,#UP_OUT                  ; Output a high pulse
                inc     UP_TEMP                     ; Increase measured duty cycle
UpPotDone:
                inc     POT_COUNT                   ; Increment the total period for
                jr      nz, GoTimer                 ; duty cycle measurement
                rcf                                 ; Divide the pot values by two to obtain
                rrc     UP_TEMP                     ; a 64-level force range
                rcf                                 ;
                rrc     DN_TEMP                     ;
                di                                  ; Subtract from 63 to reverse the direction
                ld      UPFORCE, #63          ; Calculate pot. values every 255
                sub     UPFORCE, UP_TEMP            ; counts
                ld      DNFORCE, #63          ;
                sub     DNFORCE, DN_TEMP            ;
                ei                                  ;
                clr     UP_TEMP                     ; counts
                clr     DN_TEMP                     ;
 GoTimer:
                srp     #LEARNEE_GRP          ; set the register pointer
                dec     AOBSTEST                    ; decrease the aobs test timer
                jr      nz,NOFAIL                   ; if the timer not at 0 then it didnot fail
                ld      AOBSTEST,#11          ; if it failed reset the timer
                tm      AOBSF,#00100000b            ; If the aobs was blocked before,
                jr      nz, BlockedBeam             ;   don't turn on the light
                or      AOBSF,#10000000b            ; Set the break edge flag
 BlockedBeam:
                or      AOBSF,#00100000b            ; Set the single break flag
 NOFAIL:
                inc     RadioTimeOut
                cp      OBS_COUNT, #00              ; Test for protector timed out
                jr      z, TEST125                  ; If it has failed, then don't decrement
```

```
          dec     OBS_COUNT                   ; Decrement the timer

PPointDeb:
          di                                  ; Disable ints while debouncer being modified (16us)
          tm      PPointPort, #PassPoint      ; Test for pass point being seen
          jr      nz, IncPPDeb        ; If high, increment the debouncer
DecPPDeb:
          and     PPOINT_DEB, #00000011b      ; Debounce 3-0
          jr      z, PPDebDone        ; If already zero, don't decrement
          dec     PPOINT_DEB                  ; Decrement the debouncer
          jr      PPDebDone                   ;
IncPPDeb:
          inc     PPOINT_DEB                  ; Increment 0-3 debouncer
          and     PPOINT_DEB, #00000011B      ;
          jr      nz, PPDebDone       ; If rolled over,
          ld      PPOINT_DEB, #00000011B      ; keep it at the max.
PPDebDone:
          ei                                  ; Re-enable interrupts
TEST125:
          inc     t125ms                      ; increment the 125 mS timer
          cp      t125ms,#125                 ; test for the time out
          jr      z,ONE25MS                   ; if true the jump
          cp      t125ms,#63                  ; test for the other timeout
          jr      nz,N125
          call    FAULTB
N125:
          pop     RP
          iret
ONE25MS:
          cp      RsMode, #00                 ; Test for not in RS232 mode
          jr      z, CheckSpeed       ; If not, don't update RS timer
          dec     RsMode              ; Count down RS232 time
          jr      nz, CheckSpeed              ; If not done yet, don't clear wall
          ld      STATUS, #CHARGE             ; Revert to charging wall control
CheckSpeed:
          cp      RampFlag, #STILL            ; Test for still motor
          jr      z, StopMotor        ; If so, turn off the FET's
          tm      BLINK_HI, #10000000b        ; If we are flashing the warning light,
          jr      z, StopMotor        ; then don't ramp up the motor
          cp      L_A_C, #076H        ; Special case -- use the ramp-down
          jr      z, NormalRampFlag           ; when we're going to the learned up limit
          cp      L_A_C, #070H        ; If we're learning limits,
          jr      uge, RunReduced             ; then run at a slow speed
NormalRampFlag:
          cp      RampFlag, #RAMPDOWN ; Test for slowing down
          jr      z, SlowDown                 ; If so, slow to minimum speed
SpeedUp:
          cp      PowerLevel, MaxSpeed        ; Test for at max. speed
          jr      uge, SetAtFull              ; If so, leave the duty cycle alone
RampSpeedUp:
          inc     PowerLevel                  ; Increase the duty cycle of the phase
          jr      SpeedDone                   ;
SlowDown:
          cp      PowerLevel, MinSpeed        ; Test for at min. speed
          jr      ult, RampSpeedUp            ; If we're below the minimum, ramp up to it
          jr      z, SpeedDone        ; If we're at the minimum, stay there
          dec     PowerLevel                  ; Increase the duty cycle of the phase
          jr      SpeedDone                   ;
RunReduced:
          ld      RampFlag, #FULLSPEED        ; Flag that we're not ramping up
          cp      MinSpeed, #8        ; Test for high minimum speed
          jr      ugt, PowerAtMin             ;
          ld      PowerLevel, #8              ; Set the speed at 40%
          jr      SpeedDone                   ;
PowerAtMin:
          ld      PowerLevel, MinSpeed        ; Set power at higher minimum
          jr      SpeedDone                   ;

StopMotor:
```

```
            clr     PowerLevel                  ; Make sure that the motor is stopped (FMEA
protection)
            jr      SpeedDone                   ;
SetAtFull:
            ld      RampFlag, #FULLSPEED        ; Set flag for done with ramp-up
SpeedDone:
            cp      LinePer, #36        ; Test for 50Hz or 60Hz
            jr      uge, FiftySpeed             ; Load the proper table
SixtySpeed:
            di                                  ; Disable interrupts to avoid pointer collizion
            srp     #RadioGroup                 ; Use the radio pointers to do a ROM fetch
            ld      pointerh, #HIGH(SPEED_TABLE_60)  ; Point to the force look-up table
            ld      pointerl, #LOW(SPEED_TABLE_60)   ;
            add     pointerl, PowerLevel                    ; Offset for current phase step
            adc     pointerh, #00H                          ;
            ldc     addvalueh, @pointer             ; Fetch the ROM data for phase control
            ld      PhaseTime, addvalueh                    ; Transfer to the proper register
            ei                                  ; Re-enable interrupts
            jr      WorkCheck                   ; Check the worklight toggle

FiftySpeed:
            di                                  ; Disable interrupts to avoid pointer collision
            srp     #RadioGroup                 ; Use the radio pointers to do a ROM fetch
            ld      pointerh, #HIGH(SPEED_TABLE_50)  ; Point to the force look-up table
            ld      pointerl, #LOW(SPEED_TABLE_50)   ;
            add     pointerl, PowerLevel                    ; Offset for current phase step
            adc     pointerh, #00H                          ;
            ldc     addvalueh, @pointer             ; Fetch the ROM data for phase control
            ld      PhaseTime, addvalueh                    ; Transfer to the proper register
            ei                                  ; Re-enable interrupts
WorkCheck:
            srp     #LEARNEE_GRP        ; Re-set the RP
4-22-97
            CP      EnableWorkLight,#11000000B
            JP      EQ,DontInc                  ;Has the button already been held for 10s?
            INC     EnableWorkLight             ;Work light function is added to every
                                                ;125ms if button is light button is held
                                                ;for 10s will iniate change, if not held
                                                ;down will be cleared in switch routine

DontInc:    cp      AUXLEARNSW,#0FFh            ; test for the rollover position
            jr      z,SKIPAUXLEARNSW            ; if so then skip
            inc     AUXLEARNSW          ; increase
SKIPAUXLEARNSW:
            cp      ZZWIN,#0FFH                 ; test for the roll position
            jr      z,TESTFA                    ; if so skip
            inc     ZZWIN                       ; if not increase the counter
TESTFA:
            call    FAULTB                      ; call the fault blinker
            clr     T125MS                      ; reset the timer
            inc     DOG2                        ; incrwease the second watch dog
            di
            inc     SDISABLE                    ; count off the system disable timer
            jr      nz,DO12                     ; if not rolled over then do the 1.2 sec
            dec     SDISABLE                    ; else reset to FF
DO12:
            cp      ONEP2,#00                   ; test for 0
            jr      z,INCLEARN                  ; if counted down then increment learn
            dec     ONEP2                       ; else down count
INCLEARN:
            inc     learnt                      ; increase the learn timer
            cp      learnt,#0H                  ; test for overflow
            jr      nz,LEARNTOK                 ; if not 0 skip back turning
            dec     learnt                      ;
LEARNTOK:
            ei
            inc     eraset                      ; increase the erase timer
            cp      eraset,#0H                  ; test for overflow
            jr      nz,ERASETOK                 ; if not 0 skip back turning
```

```
                dec     eraset                          ;
ERASETOK:
                pop     RP
                iret

;       fault blinker

FAULTB:
                inc     FAULTTIME                       ; increase the fault timer
                cp      L_A_C, #070H            ; Test for in learn limits mode
                jr      ult, DoFaults           ; If not, handle faults normally
                cp      L_A_C, #071H            ; Test for failed learn
                jr      z, FastFlash            ; If so, blink the LED fast
RegFlash:
                tm      FAULTTIME, #00000100b           ; Toggle the LED every 250ms
                jr      z, FlashOn                      ;
FlashOff:
                or      ledport, #ledh                  ; Turn off the LED for blink
                jr      NOFAULT                         ; Don't test for faults
FlashOn:
                and     ledport, #ledl                  ; Turn on the LED for blink
                jr      NOFAULT                         ;
FastFlash:
                tm      FAULTTIME, #00000010b           ; Toggle the LED every 125ms
                jr      z, FlashOn                      ;
                jr      FlashOff                        ;
DoFaults:
                cp      FAULTTIME,#80h                  ; test for the end
                jr      nz,FIRSTFAULT                   ; if not timed out
                clr     FAULTTIME                       ; reset the clock
                clr     FAULT                           ; clear the last
                cp      FAULTCODE,#05h                  ; test for call dealer code
                jr      UGE,GOTFAULT            ; set the fault
                cp      CMD_DEB,#0FFH                   ; test the debouncer
                jr      nz,TESTAOBSM            ; if not set test aobs
                cp      FAULTCODE,#03h                  ; test for command shorted
                jr      z,GOTFAULT                      ; set the error
                ld      FAULTCODE,#03h                  ; set the code
                jr      FIRSTFAULT                      ;
TESTAOBSM:
                tm      AOBSF,#00000001b                ; test for the skiped aobs pulse
                jr      z,NOAOBSFAULT                   ; if no skips then no faults
                tm      AOBSF,#00000010b                ; test for any pulses
                jr      z,NOPULSE                       ; if no pulses find if hi or low
                                                        ; else we are intermittent
                ld      FAULTCODE,#04h                  ; set the fault
                jr      GOTFAULT                        ; if same got fault
;               cp      FAULTCODE,#04h                  ; test the last fault
;               jr      z,GOTFAULT                      ; if same got fault
;               ld      FAULTCODE,#04h                  ; set the fault
;               jr      FIRSTFC                         ;
NOPULSE:        tm      P3,#00000001b                   ; test the input pin
                jr      z,AOBSSH                        ; jump if aobs is stuck hi
                cp      FAULTCODE,#01h                  ; test for stuck low in the past
                jr      z,GOTFAULT                      ; set the fault
                ld      FAULTCODE,#01h                  ; set the fault code
                jr      FIRSTFC                         ;
AOBSSH:         cp      FAULTCODE,#02h                  ; test for stuck high in past
                jr      z,GOTFAULT                      ; set the fault
                ld      FAULTCODE,#02h                  ; set the code
                jr      FIRSTFC                         ;

GOTFAULT:       ld      FAULT,FAULTCODE                 ; set the code
                swap    FAULT                           ;
                jr      FIRSTFC                         ;
NOAOBSFAULT:
                clr     FAULTCODE                       ; clear the fault code
FIRSTFC:        and     AOBSF, #11111100b               ; clear flags
```

```
FIRSTFAULT:
            tm      FAULTTIME, #00000111b      ; If one second has passed,
            jr      nz, RegularFault           ; increment the 60min

            incw    HOUR_TIMER                 ; Increment the 1 hour timer
            tcm     HOUR_TIMER_LO, #00011111b  ; If 32 seconds have passed
            jr      nz, RegularFault                   ; poll the radio mode

            or      AOBSF, #01000000b          ; Set the 'poll radio' flag

RegularFault:

            cp      FAULT,#00                  ; test for no fault
            jr      z,NOFAULT
            ld      FAULTFLAG,#0FFH            ; set the fault flag
            cp      CodeFlag,#REGLEARN         ; test for not in learn mode
            jr      z,TESTSDI                  ; if in learn then skip setting
            cp      FAULT,FAULTTIME            ;
            jr      ULE,TESTSDI

            tm      FAULTTIME,#00001000b       ; test the 1 sec bit
            jr      nz,BITONE
            and     ledport,#ledl                      ; turn on the led
            ret

BITONE:
            or      ledport,#ledh                      ; turn off the led
TESTSDI:
            ret

NOFAULT:    clr     FAULTFLAG                  ; clear the flag
            ret

;-----------------------------------------------------------------------

;       Four ms timer tick routines and aux light function

;-----------------------------------------------------------------------

TIMER4MS:
            cp      RPMONES,#00H           ; test for the end of the one sec timer
            jr      z,TESTPERIOD           ; if one sec over then test the pulses
                                           ; over the period
            dec     RPMONES                    ; else decrease the timer
            di
            clr     RPM_COUNT                  ; start with a count of 0
            clr     BRPM_COUNT                 ; start with a count of 0
            ei
            jr      RPMTDONE
TESTPERIOD:
            cp      RPMCLEAR,#00H              ; test the clear test timer for 0
            jr      nz,RPMTDONE            ; if not timed out then skip
            ld      RPMCLEAR,#122              ; set the clear test time for next cycle .5
            cp      RPM_COUNT,#50              ; test the count for too many pulses
            jr      ugt,FAREV                  ; if too man pulses then reverse
            di
            clr     RPM_COUNT                  ; clear the counter
            clr     BRPM_COUNT                 ; clear the counter
            ei
;           clr     FAREVFLAG                  ; clear the flag    temp test
            jr      RPMTDONE                   ; continue
FAREV:
            ld      FAULTCODE,#06h             ; set the fault flag
            ld      FAREVFLAG,#088H            ; set the forced up flag
            and     p0,#LOW ~WORKLIGHT         ; turn off light
            ld      REASON,#80H            ; rpm forcing up motion
            call    SET_AREV_STATE             ; set the autorev state
RPMTDONE:
            dec     RPMCLEAR                   ; decrement the timer
```

```
                cp      LIGHT1S,#00                     ; test for the end
                jr      z,SKIPLIGHTE
                dec     LIGHT1S                         ; down count the light time
SKIPLIGHTE:
                inc     R_DEAD_TIME
                cp      RTO,#RDROPTIME                  ; test for the radio time out
                jr      ult,DONOTCB                     ; if not timed out donot clear b
                cp      CodeFlag, #LRNOCS               ; If we are in a special learn mode,
                jr      uge, DONOTCB            ; then don't clear the code flag
                clr     CodeFlag                        ; else clear the b code flag
DONOTCB:
                inc     RTO                             ; increment the radio time out
                jr      nz,RTOOK                        ; if the radio timeout ok then skip
                dec     RTO                             ; back turn
RTOOK:
                cp      RRTO,#0FFH                      ; test for roll
                jr      z,SKIPRRTO                      ; if so then skip
                inc     RRTO
SKIPRRTO:                                               ;
                cp      SKIPRADIO, #00                  ; Test for EEPROM communication
                jr      nz, LEARNDBOK           ; If so, skip reading program switch
                cp      RsMode, #00                     ; Test for in RS232 mode,
                jr      nz, LEARNDBOK           ; if so, don't update the debouncer
                tm      psport,#psmask                  ; Test for program switch
                jr      z,PRSWCLOSED            ; if the switch is closed count up
                cp      LEARNDB,#00             ; test for the non decrement point
                jr      z,LEARNDBOK             ; if at end skip dec
                dec     LEARNDB                         ;
                jr      LEARNDBOK                       ;

PRSWCLOSED:
                cp      LEARNDB,#0FFH                   ; test for debouncer at max.
                jr      z,LEARNDBOK             ; if not at max increment
                inc     LEARNDB                         ; increase the learn debounce timer
LEARNDBOK:
;-----------------------------------------------------------------------
; AUX OBSTRUCTION OUTPUT AND LIGHT FUNCTION
;-----------------------------------------------------------------------

AUXLIGHT:
test_light_on:
        cp      LIGHT_FLAG,#LIGHT                       ;
        jr      z,dec_light                             ;
        cp      LIGHT1S,#00                             ; test for no flash
        jr      z,NO1S                                  ; if not skip
        cp      LIGHT1S,#1                              ; test for timeout
        jr      nz,NO1S                                 ; if not skip
        xor     p0,#WORKLIGHT                           ; toggle light
        clr     LIGHT1S                                 ; oneshoted
NO1S:
        cp      FLASH_FLAG,#FLASH
        jr      nz,dec_light            ;
        clr     VACFLASH                                ; Keep the vacation flash timer off
        dec     FLASH_DELAY                             ; 250 ms period
        jr      nz,dec_light            ;

        cp      STATUS, #RSSTATUS                       ; Test for in RS232 mode
        jr      z, BlinkDone            ; If so, don't blink the LED
        ; Toggle the wall control LED
        cp      STATUS, #WALLOFF                        ; See if the LED is off or on
        jr      z, TurnItOn                             ;
TurnItOff:
        ld      STATUS, #WALLOFF                        ; Turn the light off
        jr      BlinkDone                               ;
TurnItOn:
        ld      STATUS, #CHARGE                         ; Turn the light on
        ld      SWITCH_DELAY, #CMD_DEL_EX  ; Reset the delay time for charge
BlinkDone:
        ld      FLASH_DELAY,#FLASH_TIME
        dec     FLASH_COUNTER                   ;
        jr      nz,dec_light
        clr     FLASH_FLAG                      ;
dec_light:
        cp      LIGHT_TIMER_HI,#0FFH            ; test for the timer ignore
        jr      z,exit_light                    ; if set then ignore
        tm      T0EXT, #00010000b               ; Decrement the light every 8 ms
        jr      nz,exit_light           ; (Use T0Ext to prescale)
        decw    LIGHT_TIMER                     ;
        jr      nz,exit_light                   ; if timer 0 turn off the light
        and     p0,#(~LIGHT_ON)                 ; turn off the light
        cp      L_A_C, #00                      ; Test for in a learn mode
        jr      z, exit_light           ; If not, leave the LED alone
        clr     L_A_C                           ; Leave the learn mode
        or      ledport,#ledh           ; turn off the LED for program mode
exit_light:
        ret                                     ; return


;-----------------------------------------------------------------------------
; MOTOR STATE MACHINE
;-----------------------------------------------------------------------------


STATEMACHINE:
        cp      MOTDEL, #0FFH           ; Test for max. motor delay
        jr      z, MOTDELDONE           ; if do, don't increment
        inc     MOTDEL                  ; update the motor delay
MOTDELDONE:
        xor     p2,#FALSEIR                     ; toggle aux output
        cp      DOG2,#8                         ; test the 2nd watchdog for problem
        jp      ugt,START                       ; if problem reset
        cp      STATE,#6                        ; test for legal number
        jp      ugt,start                       ; if not the reset
        jp      z,stop                          ; stop motor 6
        cp      STATE,#3                        ; test for legal number
        jp      z,start                                 ; if not the reset
        cp      STATE,#0                        ; test for autorev
        jp      z,auto_rev                      ; auto reversing 0
        cp      STATE,#1                        ; test for up
        jp      z,up_direction                          ; door is going up 1
        cp      STATE,#2                        ; test for autorev
        jp      z,up_position                   ; door is up 2
        cp      STATE,#4                        ; test for autorev
        jp      z,dn_direction                          ; door is going down 4
        jp      dn_position                     ; door is down        5

;-----------------------------------------------------------------------------
;       AUTO_REV ROUTINE
;-----------------------------------------------------------------------------

auto_rev:
        cp      FAREVFLAG,#088H                 ; test for the forced up flag
        jr      nz,LEAVEREV
        and     p0,#LOW(~WORKLIGHT)     ; turn off light
;       clr     FAREVFLAG                       ; one shot temp test
LEAVEREV:
        cp      MOTDEL, #10                     ; Test for 40 ms passed
        jr      ult, AREVON                     ; If not, keep the relay on
AREVOFF:
        and     p0,#LOW(~MOTOR_UP & ~MOTOR_DN)  ; disable motor
AREVON:
        WDT                                     ; kick the dog
        call    HOLDFREV                        ; hold off the force reverse
        ld      LIGHT_FLAG,#LIGHT               ; force the light on no blink
        di
        dec     AUTO_DELAY                      ; wait for .5 second
        dec     BAUTO_DELAY             ; wait for .5 second
        ei
        jr      nz,arswitch                     ; test switches

        or      p2,#FALSEIR                     ; set aux output    for FEMA

        ;LOOK FOR LIMIT HERE(No)
        ld      REASON,#40H                     ; set the reason for the change
        cp      L_A_C, #075H            ; Check for learning limits,
        jp      nz, SET_UP_NOBLINK      ; If not, proceed normally
        ld      L_A_C, #076H            ;
        jp      SET_UP_NOBLINK                  ; set the state
arswitch:
        ld      REASON,#00H                     ; set the reason to command
        di
        cp      SW_DATA,#CMD_SW                 ; test for a command
        clr     SW_DATA
        ei
        jp      z,SET_STOP_STATE                ; if so then stop
        ld      REASON,#10H                     ; set the reason as radio command
        cp      RADIO_CMD,#0AAH                 ; test for a radic command
        jp      z,SET_STOP_STATE                ; if so the stop
exit_auto_rev:
        ret                                     ; return


HOLDFREV:
        ld      RPMONES,#244            ; set the hold off
        ld      RPMCLEAR,#122                   ; clear rpm reverse .5 sec
        di
        clr     RPM_COUNT                       ; start with a count cf 0
        clr     BRPM_COUNT                      ; start with a count of 0
        ei
        ret


;------------------------------------------------------------------------
;       DOOR GOING UP
;------------------------------------------------------------------------

up_direction:
        WDT                                     ; kick the dog
        cp      OnePass, STATE                  ; Test for the memory read one-shot
        jr      z, UpReady                      ; If so, continue
        ret                                     ; Else wait
UpReady:
        call    HOLDFREV                        ; hold off the force reverse
        ld      LIGHT_FLAG,#LIGHT               ; force the light on no blink
        and     p0,#LOW ~MOTOR_DN       ; disable down relay

        or      p0,#LIGHT_ON                    ; turn on the light
        cp      MOTDEL,#10                      ; test for 40 milliseconds
        jr      ule,UPOFF                       ; if not timed
CheckUpBlink:
        and     P2M_SHADOW, #~BLINK_PIN         ; Turn on the blink output
        ld      P2M, P2M_SHADOW                 ;
        or      P2, #BLINK_PIN                  ; Turn on the blinker
        decw    BLINK                           ; Decrement blink time
        tm      BLINK_HI, #10000000b            ; Test for pre-travel blinking done
        jp      z, NotUpSlow            ; If not, delay normal motor travel

UPON:
        or      p0,#(MOTOR_UP | LIGHT_ON)  ; turn on the motor and light
UPOFF:
        cp      FORCE_IGNORE,#1                 ; test fro the end of the force ignore
        jr      nz,SKIPUPRPM            ; if not donot test rpmcount
        cp      RPM_ACOUNT,#02H                 ; test for less the 2 pulses
        jr      ugt,SKIPUPRPM                   ;
        ld      FAULTCODE,#05h
SKIPUPRPM:
        cp      FORCE_IGNORE,#00                ; test timer for done
        jr      nz,test_up_sw_pre               ; if timer not up do not test force
TEST_UP_FORCE:
        di
        dec     RPM_TIME_OUT            ; decrease the timeout
        dec     BRPM_TIME_OUT                   ; decrease the timeout
        ei
        jr      z,failed_up_rpm
        cp      RampFlag, #RAMPUP               ; Check for ramping up the force
        jr      z, test_up_sw           ; If not, always do full force check
TestUpForcePot:
        di                                      ; turn off the interrupt
        cp      RPM_PERIOD_HI, UP_FORCE_HI ; Test the RPM against the force setting
        jr      ugt, failed_up_rpm      ;
        jr      ult, test_up_sw                 ;
        cp      RPM_PERIOD_LO, UP_FORCE_LO ;
        jr      ult, test_up_sw                 ;
failed_up_rpm:
        ld      REASON,#20H                     ; set the reason as force
        cp      L_A_C, #076H            ; If we're learning limits,
        jp      nz, SET_STOP_STATE      ; then set the flag to store
        ld      L_A_C, #077H            ;
        jp      SET_STOP_STATE
test_up_sw_pre:
        di
        dec     FORCE_IGNORE
        dec     BFORCE_IGNORE
test_up_sw:
        di
        ld      LIM_TEST_HI, POSITION_HI   ; Calculate the distance from the up limit
        ld      LIM_TEST_LO, POSITION_LO   ;
        sub     LIM_TEST_LO, UP_LIMIT_LO   ;
        sbc     LIM_TEST_HI, UP_LIMIT_HI   ;
        cp      POSITION_HI, #0B0H         ; Test for lost door
        jr      ugt, UpPosKnown                 ; If not lost, limit test is done
        cp      POSITION_HI, #050H         ;
        jr      ult, UpPosKnown                 ;
        ei                                      ;
UpPosUnknown:
        sub     LIM_TEST_LO, #062H         ; Calculate the total travel distance allowed
        sbc     LIM_TEST_HI, #07FH         ; from the floor when lost
        add     LIM_TEST_LO, DN_LIMIT_LO   ;
        adc     LIM_TEST_HI, DN_LIMIT_HI   ;
UpPosKnown:                                     ;
        ei                                      ;
        cp      L_A_C, #070H               ; If we're positioning the door, forget the limit
        jr      z, test_up_time                 ; and the wall control and radio
        cp      LIM_TEST_HI, #00                ; Test for exactly at the limit
        jr      nz, TestForPastUp               ; If not, see if we've passed the limit
        cp      LIM_TEST_LO, #00                ;
        jr      z, AtUpLimit               ;
TestForPastUp:
        tm      LIM_TEST_HI, #10000000b         ; Test for a negative result (past the limit, but
close)
        jr      z, get_sw                       ; If so, set the limit
AtUpLimit:
        ld      REASON,#50H                     ; set the reason as limit
        cp      L_A_C, #072H               ; If we're re-learning limits,
        jr      z, ReLearnLim              ; jump
        cp      L_A_C, #076H               ; If we're learning limits,
        jp      nz, SET_UP_POS_STATE            ; then set the flag to store
        ld      L_A_C, #077H               ;
        jp      SET_UP_POS_STATE                ;
ReLearnLim:
        ld      L_A_C, #073H               ;
        jp      SET_UP_POS_STATE                ;
get_sw:
        cp      L_A_C, #070H               ; Test for positioning the up limit
        jr      z,NotUpSlow                     ; If so, don't slow down
```

```
TestUpSlow:
        cp      LIM_TEST_HI, #HIGH(UPSLOWSTART)   ; Test for start of slowdown
        jr      nz, NotUpSlow             ;  (Cheating -- the high byte of the number is zero)
        cp      LIM_TEST_LO, #LOW(UPSLOWSTART)    ;
        jr      ugt, NotUpSlow                    ;
UpSlow:
        ld      RampFlag, #RAMPDOWN       ; Set the slowdown flag
NotUpSlow:
        ld      REASON,#10H                       ; set the radio command reason
        cp      RADIO_CMD,#0AAH                   ; test for a radio command
        jp      z,SET_STOP_STATE                  ; if so stop
        ld      REASON,#00H                       ; set the reason as a command
        di
        cp      SW_DATA,#CMD_SW                   ; test for a command condition
        clr     SW_DATA
        ei
        jr      ne,test_up_time                           ;
        jp      SET_STOP_STATE                    ;
test_up_time:
        ld      REASON,#70H                       ; set the reason as a time out
        decw    MOTOR_TIMER               ; decrement motor timer
        jp      z,SET_STOP_STATE                  ;
exit_up_dir:
        ret                                       ; return to caller
;-----------------------------------------------------------------------
;       DOOR UP
;-----------------------------------------------------------------------

up_position:
        WDT                                       ; kick the dog
        cp      FAREVFLAG,#088H                   ; test for the forced up flag
        jr      nz,LEAVELIGHT
        and     p0,#LOW(~WORKLIGHT)       ; turn off light
        jr      UPNOFLASH                         ; skip clearing the flash flag
LEAVELIGHT:
        ld      LIGHT_FLAG,#00H                   ; allow blink
UPNOFLASH:
        cp      MOTDEL, #10                       ; Test for 40 ms passed
        jr      ult, UPLIMON              ; If not, keep the relay on
UPLIMOFF:
        and     p0,#LOW(~MOTOR_UP & ~MOTOR_DN)    ; disable motor
UPLIMON:
        cp      L_A_C, #073H              ; If we've begun the learn limits cycle,
        jr      z,LACUPPOS                        ; then delay before traveling
        cp      SW_DATA,#LIGHT_SW         ; light sw debounced?
        jr      z,work_up                         ;
        ld      REASON,#10H                       ; set the reason as a radio command
        cp      RADIO_CMD,#0AAH                   ; test for a radio cmd
        jr      z,SETDNDIRSTATE                   ; if so start down
        ld      REASON,#00H                       ; set the reason as a command
        di
        cp      SW_DATA,#CMD_SW                   ; command sw debounced?
        clr     SW_DATA
        ei
        jr      z,SETDNDIRSTATE                   ; if command
        ret
SETDNDIRSTATE:
        ld      ONEP2,#10                         ; set the 1.2 sec timer
        jp      SET_DN_DIR_STATE

LACUPPOS:
        cp      MOTOR_TIMER_HI, #HIGH LACTIME ; Make sure we're set to the proper time
        jr      ule, UpTimeOk
        ld      MOTOR_TIMER_HI, #HIGH(LACTIME)
        ld      MOTOR_TIMER_LO, #LOW(LACTIME)
UpTimeOk:
        decw    MOTOR_TIMER                       ; Count down more time
        jr      nz, up_pos_ret                    ; If not timed out, leave
StartLACDown:
        ld      L_A_C, #074H                ; Set state as traveling down in LAC
        clr     UP_LIMIT_HI                         ; Clear the up limit
        clr     UP_LIMIT_LO                         ; and the position for
        clr     POSITION_HI                         ; determining the new up
        clr     POSITION_LO                         ; limit of travel
        ld      PassCounter, #030H          ; Set pass points at max.
        jp      SET_DN_DIR_STATE                    ; Start door traveling down

work_up:
        xor     p0,#WORKLIGHT                       ; toggle work light
        ld      LIGHT_TIMER_HI,#0FFH                ; set the timer ignore
        and     SW_DATA, #LOW(~LIGHT_SW)    ; Clear the worklight bit
up_pos_ret:
        ret                                         ; return
;---------------------------------------------------------------------------
;       DOOR GOING DOWN
;---------------------------------------------------------------------------

dn_direction:
        WDT                                         ; kick the dog
        cp      OnePass, STATE                      ; Test for the memory read one-shot
        jr      z, DownReady                ; If so, continue
        ret                                         ; else wait
DownReady:
        call    HOLDFREV                            ; hold off the force reverse
        clr     FLASH_FLAG                          ; turn off the flash
        ld      LIGHT_FLAG,#LIGHT                   ; force the light on no blink
        and     p0,#LOW(~MOTOR_UP)          ; turn off motor up

        or      p0,#LIGHT_ON                        ; turn on the light
        cp      MOTDEL,#10                          ; test for 40 milliseconds
        jr      ule,DNOFF                           ; if not timed

CheckDnBlink:
        and     P2M_SHADOW, #~BLINK_PIN             ; Turn on the blink output
        ld      P2M, P2M_SHADOW                     ;
        or      P2, #BLINK_PIN                      ; Turn on the blinker
        decw    BLINK                               ; Decrement blink time
        tm      BLINK_HI, #10000000b                ; Test for pre-travel blink done
        jr      z, NotDnSlow                ; If not, don't start the motor

DNON:

        or      p0,#(MOTOR_DN   LIGHT_ON)   ; turn on the motor and light
DNOFF:
        cp      FORCE_IGNORE,#01                    ; test fro the end of the force ignore
        jr      nz,SKIPDNRPM                ; if not donot test rpmcount
        cp      RPM_ACOUNT,#02H                     ; test for less the 2 pulses
        jr      ugt,SKIPDNRPM                       ;
        ld      FAULTCODE,#05h
SKIPDNRPM:
        cp      FORCE_IGNORE,#00                    ; test timer for done
        jr      nz,test_dn_sw_pre                   ; if timer not up do not test force

TEST_DOWN_FORCE:
        di
        dec     RPM_TIME_OUT                ; decrease the timeout
        dec     BRPM_TIME_OUT                       ; decrease the timeout
        ei
        jr      z,failed_dn_rpm
        cp      RampFlag, #RAMPUP                   ; Check for ramping up the force
        jr      z, test_dn_sw               ; If not, always do full force check
TestDownForcePot:
        di                                          ; turn off the interrupt
        cp      RPM_PERIOD_HI, DN_FORCE_HI ; Test the RPM against the force setting
        jr      ugt, failed_dn_rpm          ; if too slow then force reverse
        jr      ult, test_dn_sw                     ; if faster then we're fine
        cp      RPM_PERIOD_LO, DN_FORCE_LO ;
        jr      ult, test_dn_sw                     ;
```

```
failed_dn_rpm:
        cp      L_A_C, #074H            ; Test for learning limits
        jp      z, DnLearnRev           ; If not, set the state normally
        tm      POSITION_HI, #11000000b         ; Test for below last pass point
        jr      nz, DnRPMRev            ; if not, we're nowhere near the limit
        tm      LIM_TEST_HI, #10000000b         ; Test for beyond the down limit
        jr      nz, DoDownLimit                 ; If so, we've driven into the down limit
DnRPMRev:
        ld      REASON,#20H                     ; set the reason as force
        cp      POSITION_HI, #0B0H      ; Test for lost,
        jp      ugt, SET_AREV_STATE     ; if not, autoreverse normally
        cp      POSITION_HI, #050H      ;
        jp      ult, SET_AREV_STATE     ;
        di                                      ; Disable interrupts
        ld      POSITION_HI, #07FH      ; Reset lost position for max. travel up
        ld      POSITION_LO, #080H      ;
        ei                                      ; Re-enable interrupts
        jp      SET_AREV_STATE                  ;
DnLearnRev:
        ld      L_A_C, #075H            ; Set proper LAC
        jp      SET_AREV_STATE                  ;


test_dn_sw_pre:
        di
        dec     FORCE_IGNORE
        dec     BFORCE_IGNORE
test_dn_sw:
        di                                      ;
        cp      POSITION_HI, #050H      ; Test for lost in mid travel
        jr      ult, TestDnLimGood      ;
        cp      POSITION_HI, #0B0H      ; If so, don't test for limit until
        jr      ult, NotDnSlow                  ; a proper pass point is seen
TestDnLimGood:
        ld      LIM_TEST_HI, DN_LIMIT_HI  ; Measure the distance to the down limit
        ld      LIM_TEST_LO, DN_LIMIT_LO  ;
        sub     LIM_TEST_LO, POSITION_LO  ;
        sbc     LIM_TEST_HI, POSITION_HI  ;
        ei                                      ;

        cp      L_A_C, #070H            ; If we're in the learn cycle, forget the limit
        jr      uge, test_dn_time               ; and ignore the radio and wall control
        tm      LIM_TEST_HI, #10000000b         ; Test for a negative result (past the down limit
        jr      z, call_sw_dn           ; If so, set the limit
        cp      LIM_TEST_LO, #(255 - 36   ; Test for 36 pulses (3") beyond the limit
        jr      ugt, NotDnSlow                  ; if not, then keep driving into the floor
DoDownLimit:
        ld      REASON,#50H                     ; set the reason as a limit
        cp      CMD_DEB,#0FFH                   ; test for the switch still held
        jr      nz,TESTRADIO            ;
        ld      REASON,#90H                     ; closed with the control held
        jr      TESTFORCEIG
TESTRADIO:
        cp      LAST_CMD,#00            ; test for the last command being radio
        jr      nz,TESTFORCEIG                  ; if not test force
        cp      CodeFlag,#BRECEIVED             ; test for the b code flag
        jr      nz,TESTFORCEIG                  ;
        ld      REASON,#0A0H            ; set the reason as b code to limit
TESTFORCEIG:
        cp      FORCE_IGNORE,#00H       ; test the force ignore for done
        jr      z,NOAREVDN                      ; a rev if limit before force enabled
        ld      REASON,#60h                     ; early limit
        jp      SET_AREV_STATE                  ; set autoreverse
NOAREVDN:
        and     p0,#LOW(~MOTOR_DN)      ;
        jp      SET_DN_POS_STATE                ; set the state
call_sw_dn:
        cp      LIM_TEST_HI, #HIGH(DNSLOWSTART)  ; Test for start of slowdown
        jr      nz, NotDnSlow           ;  (Cheating -- the high byte is zero)
        cp      LIM_TEST_LO, #LOW(DNSLOWSTART)  ;
        jr      ugt, NotDnSlow                  ;
DnSlow:
        ld      RampFlag, #RAMPDOWN     ; Set the slowdown flag
NotDnSlow:
        ld      REASON,#10H                     ; set the reason as radio command
        cp      RADIO_CMD,#0AAH                 ; test for a radio command
        jp      z,SET_AREV_STATE                ; if so arev
        ld      REASON,#00H                     ; set the reason as command
        di
        cp      SW_DATA,#CMD_SW                 ; test for command
        clr     SW_DATA
        ei
        jp      z,SET_AREV_STATE                ;
test_dn_time:
        ld      REASON,#70H                     ; set the reason as timeout
        decw    MOTOR_TIMER             ; decrement motor timer
        jp      z,SET_AREV_STATE                ;
test_obs_count:
        cp      OBS_COUNT,#00                   ; Test the obs count
        jr      nz, exit_dn_dir                 ; if not done, don't reverse
        cp      FORCE_IGNORE, #(ONE_SEC / 2)    ; Test for 0.5 second passed
        jr      ugt, exit_dn_dir                ; if within first 0.5 sec, ignore it
        cp      LAST_CMD,#00            ; test for the last command from radio
        jr      z,OBSTESTB                      ; if last command was a radio test b
        cp      CMD_DEB,#0FFH                   ; test for the command switch holding
        jr      nz,OBSAREV                      ; if the command switch is not holding
                                                ; do the autorev
        jr      exit_dn_dir                     ; otherwise skip
OBSAREV:
        ld      FLASH_FLAG,#0FFH                ; set flag
        ld      FLASH_COUNTER,#20       ; set for 10 flashes
        ld      FLASH_DELAY,#FLASH_TIME         ; set for .5 Hz period
        ld      REASON,#30H                     ; set the reason as autoreverse
        jp      SET_AREV_STATE                  ;
OBSTESTB:
        cp      CodeFlag,#BRECEIVED                     ; test for the b code flag
        jr      nz,OBSAREV                      ; if not b code then arev
exit_dn_dir:
        ret                                     ; return

;-----------------------------------------------------------------------------
;       DOOR DOWN
;-----------------------------------------------------------------------------

dn_position:
        WDT                                     ; kick the dog
;       cp      FAREVFLAG,#088H                 ; test for the forced up flag
;       jr      nz,DNLEAVEL                     ;
;       and     p0,#LOW(~WORKLIGHT)     ; turn off light
;       jr      DNNOFLASH                       ; skip clearing the flash flag

DNLEAVEL:
        ld      LIGHT_FLAG,#00H                 ; allow blink
DNNOFLASH:
        cp      MOTDEL, #10                     ; Test for 40 ms passed
        jr      ult, DNLIMON            ; If not, keep the relay on
DNLIMOFF:
        and     p0,#LOW(~MOTOR_UP & ~MOTOR_DN)  ; disable motor
DNLIMON:
        cp      SW_DATA,#LIGHT_SW       ; debounced? light
        jr      z,work_dn                       ;
        ld      REASON,#10H                     ; set the reason as a radio command
        cp      RADIO_CMD,#0AAH                 ; test for a radio command
        jr      z,SETUPDIRSTATE                 ; if so go up
        ld      REASON,#00H                     ; set the reason as a command
        di
        cp      SW_DATA,#CMD_SW                 ; command sw pressed?
```

```
      clr     SW_DATA
      ei
      jr      z,SETUPDIRSTATE                   ; if so go up
      ret

SETUPDIRSTATE:
      ld      ONEP2,#10                         ; set the 1.2 sec timer
      jp      SET_UP_DIR_STATE

work_dn:
      xor     p0,#WORKLIGHT                     ; toggle work light
      ld      LIGHT_TIMER_HI,#0FFH              ; set the timer ignore
      and     SW_DATA, # LOW(~LIGHT_SW)   ; Clear the worklight bit
dn_pos_ret:
      ret                                       ; return
;-----------------------------------------------------------------------
;       STOP
;-----------------------------------------------------------------------

stop:
      WDT                                       ; kick the dog
      cp      FAREVFLAG,#088H                   ; test for the forced up flag
      jr      nz,LEAVESTOP
      and     p0,#LOW ~WORKLIGHT          ; turn off light
      jr      STOPNOFLASH                       ;
LEAVESTOP:
      ld      LIGHT_FLAG,#00H                   ; allow blink
STOPNOFLASH:
      cp      MOTDEL, #10                       ; Test for 40 ms passed
      jr      ult, STOPMIDON                    ; If not, keep the relay on
STOPMIDOFF:
      and     p0,#LOW(~MOTOR_UP & ~MOTOR_DN)    ; disable motor
STOPMIDON:
      cp      SW_DATA,#LIGHT_SW           ; debounced? light
      jr      z,work_stop                       ;
      ld      REASON,#10H                       ; set the reason as radio command
      cp      RADIO_CMD,#0AAH                   ; test for a radio command
      jp      z,SET_DN_DIR_STATE          ; if so go down
      ld      REASON,#00H                       ; set the reason as a command
      di
      cp      SW_DATA,#CMD_SW                   ; command sw pressed?
      clr     SW_DATA
      ei
      jp      z,SET_DN_DIR_STATE          ; if so go down
      ret
work_stop:
      xor     p0,#WORKLIGHT                     ; toggle work light
      ld      LIGHT_TIMER_HI,#0FFH              ; set the timer ignore
      and     SW_DATA, #LOW(~LIGHT_SW     ; Clear the worklight bit
stop_ret:
      ret                                       ; return

;-----------------------------------------------------------------------
;       SET THE AUTOREV STATE
;-----------------------------------------------------------------------
SET_AREV_STATE:
      di                                        ;
      cp      L_A_C, #070H                ; Test for learning limits,
      jr      uge, LearningRev                  ; If not, do a normal autoreverse

      cp      POSITION_HI, #020H          ; Look for lost postion
      jr      ult, DoTheArev                    ; If not, proceed as normal
      cp      POSITION_HI, #0D0H          ; Look for lost postion
      jr      ugt, DoTheArev                    ; If not, proceed as normal

      ;Otherwise, we're lost -- ignore commands
      cp      REASON, #020H               ; Don't respond to command or radio
      jr      uge, DoTheArev                    ;
      clr     RADIO_CMD                         ; Throw out the radio command
```

```
        ei                                      ; Otherwise, just ignore it
        ret                                     ;

DoTheArev:

        ld      STATE,#AUTO_REV                 ; if we got here, then reverse motor
        ld      RampFlag, #STILL                ; Set the FET's to off
        clr     PowerLevel                      ;
        jr      SET_ANY                         ; Done

LearningRev:
        ld      STATE,#AUTO_REV                 ; if we got here, then reverse motor
        ld      RampFlag, #STILL                ; Set the FET's to off
        clr     PowerLevel                      ;
        cp      L_A_C, #075H            ; Check for proper reversal
        jr      nz, ErrorLearnArev      ; If not, stop the learn cycle
        cp      PassCounter, #030H      ; If we haven't seen a pass point,
        jr      z, ErrorLearnArev               ; then flag an error
GoodLearnArev:
        cp      POSITION_HI, #00                ; Test for down limit at least
        jr      nz, DnLimGood           ; 20 pulses away from pass point
        cp      POSITION_LO, #20                ;
        jr      ult, MovePassPoint      ; If not, use the upper pass point
DnLimGood:
        and     PassCounter, #10000000b         ; Set at lowest pass point
GotDnLim:
        di
        ld      DN_LIMIT_HI, POSITION_HI   ; Set the new down limit
        ld      DN_LIMIT_LO, POSITION_LO   ;
        add     DN_LIMIT_LO, #01                ; Add in a pulse to guarantee reversal off the block
        adc     DN_LIMIT_HI, #00                ;
        jr      SET_ANY                         ;
ErrorLearnArev:
        ld      L_A_C, #071H            ; Set the error in learning state
        jr      SET_ANY

MovePassPoint:
        cp      PassCounter, #02FH            ; If we have only one pass point,
        jr      z, ErrorLearnArev                     ; don't allow it to be this close to the floor
        di
        add     POSITION_LO, #LOW(PPOINTPULSES)  ; Use the next pass point up
        adc     POSITION_HI, #HIGH(PPOINTPULSES) ;
        add     UP_LIMIT_LO, #LOW(PPOINTPULSES)  ;
        adc     UP_LIMIT_HI, #HIGH(PPOINTPULSES) ;
        ei                                           ;
        or      PassCounter, #01111111b              ; Set pass counter at -1
        jr      GotDnLim                             ;


;-----------------------------------------------------------------------
;       SET THE STOPPED STATE
;-----------------------------------------------------------------------
SET_STOP_STATE:
        di
        cp      L_A_C, #070H            ; If we're in the learn mode,
        jr      uge, DoTheStop                  ; Then don't ignore anything
        cp      POSITION_HI, #020H      ; Look for lost postion
        jr      ult, DoTheStop                  ; If not, proceed as normal
        cp      POSITION_HI, #0D0H      ; Look for lost postion
        jr      ugt, DoTheStop                  ; If not, proceed as normal

        ;Otherwise, we're lost -- ignore commands
        cp      REASON, #020H           ; Don't respond to command or radio
        jr      uge, DoTheStop                  ;
        clr     RADIO_CMD                       ; Throw out the radio command
        ei                                      ; Otherwise, just ignore it
        ret                                     ;

DoTheStop:
        ld      STATE,#STOP                     ;
        ld      RampFlag, #STILL                ; Stop the motor at the FET's
        clr     PowerLevel                      ;
        jr      SET_ANY

;-----------------------------------------------------------------
;       SET THE DOWN DIRECTION STATE
;-----------------------------------------------------------------
SET_DN_DIR_STATE:

        ld      BLINK_HI, #0FFH                 ;Initially disable pre-travel blink
        call    LookForFlasher                  ;Test to see if flasher present
        tm      P2, #BLINK_PIN                  ;If the flasher is not present,
        jr      nz, SET_DN_NOBLINK      ;don't flash it
        ld      BLINK_LO, #0FFH                 ;Turn on the blink timer
        ld      BLINK_HI, #01H                  ;

SET_DN_NOBLINK:

        di
        ld      RampFlag, #RAMPUP               ; Set the flag to accelerate motor
        ld      PowerLevel, #4                  ; Set speed at minimum
        ld      STATE,#DN_DIRECTION             ; energize door
        clr     FAREVFLAG                       ; one shot the forced reverse

        cp      L_A_C, #070H            ; If we're learning the limits,
        jr      uge, SET_ANY            ; Then don't bother with testing anything

        cp      POSITION_HI, #020H      ; Look for lost postion
        jp      ult, SET_ANY            ; If not, proceed as normal
        cp      POSITION_HI, #0D0H      ; Lock for lost postion
        jp      ugt, SET_ANY            ; If not, proceed as normal

LostDn:

        cp      FirstRun, #00           ; If this isn't our first operation when lost,
        jr      nz, SET_ANY                     ; then ALWAYS head down
        tm      PassCounter, #01111111b         ; If we are below the lowest
        jr      z, SET_UP_DIR_STATE     ; pass point, head up to see it
        tcm     PassCounter, #01111111b         ; If our pass point number is set at -1,
        jr      z, SET_UP_DIR_STATE     ; then go up to find the position
        jr      SET_ANY                         ; Otherwise, proceed normally

;-----------------------------------------------------------------
;       SET THE DOWN POSITION STATE
;-----------------------------------------------------------------
SET_DN_POS_STATE:
        di
        ld      STATE,#DN_POSITION      ; load new state
        ld      RampFlag, #STILL                ; Stop the motor at the FET's
        clr     PowerLevel                      ;
        jr      SET_ANY

;-----------------------------------------------------------------
;       SET THE UP DIRECTION STATE
;-----------------------------------------------------------------
SET_UP_DIR_STATE:

        ld      BLINK_HI, #0FFH                 ;Initially turn off blink
        call    LookForFlasher                  ;Test to see if flasher present
        tm      P2, #BLINK_PIN                  ;If the flasher is not present,
        jr      nz, SET_UP_NOBLINK      ;don't flash it
        ld      BLINK_LO, #0FFH                 ;Turn on the blink timer
        ld      BLINK_HI, #01H                  ;

SET_UP_NOBLINK:
        di
        ld      RampFlag, #RAMPUP               ; Set the flag to accelerate to max.
        ld      PowerLevel, #4                  ; Start speed at minimum
```

```
       ld     STATE,#UP_DIRECTION                       ;
       jr     SET_ANY                                   ;

;-------------------------------------------------------------------------------
;      SET THE UP POSITION STATE
;-------------------------------------------------------------------------------
SET_UP_POS_STATE:
       di
       ld     STATE,#UP_POSITION                 ;
       ld     RampFlag, #STILL                          ; Stop the motor at the FET's
       clr    PowerLevel                                ;

;-------------------------------------------------------------------------------
;      SET ANY STATE
;-------------------------------------------------------------------------------
SET_ANY:
       and    P2M_SHADOW, #~BLINK_PIN                   ; Turn on the blink output
       ld     P2M, P2M_SHADOW                           ;
       and    P2, #~BLINK_PIN                           ; Turn off the light

       cp     PPOINT_DEB, #2                            ; Test for pass point being seen
       jr     ult, NoPrePPoint                          ; If signal is low, none seen
PrePPoint:
       or     PassCounter, #10000000b                   ; Flag pass point signal high
       jr     PrePPointDone                      ;
NoPrePPoint:
       and    PassCounter, #01111111b                   ; Flag pass point signal low
PrePPointDone:

;      ld     FirstRun, #0FFH                           ; One-shot the first run flag DONE IN MAIN
       ld     BSTATE,STATE                       ; set the backup state
       di
       clr    RPM_COUNT                                 ; clear the rpm counter
       clr    BRPM_COUNT                                ;
       ld     AUTO_DELAY,#AUTO_REV_TIME          ; set the .5 second auto rev timer
       ld     BAUTO_DELAY,#AUTO_REV_TIME         ;
       ld     FORCE_IGNORE,#ONE_SEC                     ; set the force ignore timer to one sec
       ld     BFORCE_IGNORE,#ONE_SEC                    ; set the force ignore timer to one sec
       ld     RPM_PERIOD_HI, #0FFH                      ; Set the RPM period to max. to start
       ei                                               ; Flush out any pending interrupts
       di                                               ;
       cp     L_A_C, #070H                       ; If we are in learn mode,
       jr     uge, LearnModeMotor                ; don't test the travel distance
       push   LIM_TEST_HI                               ; Save the limit tests
       push   LIM_TEST_LO                               ;
       ld     LIM_TEST_HI, DN_LIMIT_HI           ; Test the door travel distance to
       ld     LIM_TEST_LO, DN_LIMIT_LO           ; see if we are shorter than 2.3M
       sub    LIM_TEST_LO, UP_LIMIT_LO           ;
       sbc    LIM_TEST_HI, UP_LIMIT_HI           ;
       cp     LIM_TEST_HI, #HIGH(SHORTDOOR)             ; If we are shorter than 2.3M,
       jr     ugt, DoorIsNorm                           ; then set the max. travel speed to 2/3
       jr     ult, DoorIsShort                          ; Else, normal speed
       cp     LIM_TEST_LO, #LOW(SHORTDOOR)              ;
       jr     ugt, DoorIsNorm                           ;
DoorIsShort:
       ld     MaxSpeed, #12                      ; Set the max. speed to 2/3
       jr     DoorSet                                   ;
DoorIsNorm:
       ld     MaxSpeed, #20                      ;
DoorSet:
       pop    LIM_TEST_LO                               ; Restore the limit tests
       pop    LIM_TEST_HI                               ;
       ld     MOTOR_TIMER_HI,#HIGH(MOTORTIME)
       ld     MOTOR_TIMER_LO,#LOW(MOTORTIME)
MotorTimeSet:
       ei
       clr    RADIO_CMD                                 ; one shot
       clr    RPM_ACOUNT                                ; clear the rpm active counter
       ld     STACKREASON,REASON                 ; save the temp reason
```

```
        ld      STACKFLAG,#0FFH                 ; set the flag
TURN_ON_LIGHT:
        call    SetVarLight                     ; Set the worklight to the proper value
        tm      P0, #LIGHT_ON           ; If the light is on skip clearing
        jr      nz,lighton                      ;
lightoff:
        clr     MOTDEL                  ; clear the motor delay
lighton:
        ret

LearnModeMotor:
        ld      MaxSpeed, #12           ; Default to slower max. speed
        ld      MOTOR_TIMER_HI,#HIGH(LEARNTIME)
        ld      MOTOR_TIMER_LO,#LOW(LEARNTIME)
        jr      MotorTimeSet            ; Set door to longer run for learn

;---------------------------------------------------------------------------
;       THIS IS THE MOTOR RPM INTERRUPT ROUTINE
;---------------------------------------------------------------------------
RPM:
        push    rp                              ; save current pointer
        srp     #RPM_GROUP                      ;point to these reg
        ld      rpm_temp_of,T0_OFLOW            ; Read the 2nd extension
        ld      rpm_temp_hi,T0EXT               ; read the timer extension
        ld      rpm_temp_lo,T0                  ; read the timer
        tm      IRQ,#00010000B                  ; test for a pending interrupt
        jr      z,RPMTIMEOK                     ; if not then time ok
RPMTIMEERROR:
        tm      rpm_temp_lo,#10000000B          ; test for timer reload
        jr      z,RPMTIMEOK                     ; if no reload time is ok
        decw    rpm_temp_hiword                 ; if reloaded then dec the hi to resync
RPMTIMEOK:
        cp      RPM_FILTER, #128                ; Signal must have been high for 3 ms before
        jr      ult, RejectTheRPM               ; the pulse is considered legal
        tm      P3, #00000010B                  ; If the line is sitting high,
        jr      nz, RejectTheRPM                ; then the falling edge was a noise pulse

RPMIsGood:
        and     imr,#11111011b                  ; turn off the interupt for up to 500uS

        ld      divcounter, #03                 ; Set to divide by 8 (destroys value in RPM_FILTER)
DivideRPMLoop:
        rcf                                     ; Reset the carry
        rrc     rpm_temp_of                     ; Divide the number by 8 so that
        rrc     rpm_temp_hi                     ; it will always fit within 16 bits
        rrc     rpm_temp_lo                     ;
        djnz    divcounter, DivideRPMLoop ; Loop three times (Note:  This clears RPM_FILTER)

        ld      rpm_period_lo, rpm_past_lo ;
        ld      rpm_period_hi, rpm_past_hi ;
        sub     rpm_period_lo, rpm_temp_lo ; find the period of the last pulse
        sbc     rpm_period_hi, rpm_temp_hi ;

        ld      rpm_past_lo, rpm_temp_lo   ; Store the current time for the
        ld      rpm_past_hi, rpm_temp_hi   ; next edge capture

        cp      rpm_period_hi,#12          ; test for a period of at least 6.144mS
        jr      ult,SKIPC                       ; if the period is less then skip counting
TULS:
INCRPM:
        inc     RPM_COUNT                       ; increase the rpm count
        inc     BRPM_COUNT                      ; increase the rpm count
SKIPC:
        inc     RPM_ACOUNT                      ; increase the rpm count
        cp      RampFlag, #RAMPUP               ; If we're ramping the speed up,
        jr      z, MaxTimeOut              ; then set the timeout at max.
        cp      STATE, #DN_DIRECTION            ; If we're traveling down,
        jr      z, DownTimeOut                  ; then set the timeout from the down force
UpTimeOut:
        ld      rpm_time_out,UP_FORCE_HI   ; Set the RPM timeout to be equal to the up force setting
        rcf                                        ; Divide by two to account
        rrc     rpm_time_out               ; for the different prescalers
        add     rpm_time_out, #2                   ; Round up and account for free-running prescale
        jr      GotTimeOut
MaxTimeOut:
        ld      rpm_time_out, #125         ; Set the RPM timeout to be 500ms
        jr      GotTimeOut                         ;
DownTimeOut:
        ld      rpm_time_out,DN_FORCE_HI   ; Set the RPM timeout to be equal to the down force setting
        rcf                                        ; Divide by two to account
        rrc     rpm_time_out               ; for the different prescalers
        add     rpm_time_out, #2                   ; Round up and account for free-running prescale
GotTimeOut:
        ld      BRPM_TIME_OUT,rpm_time_out ; Set the backup to the same value
        ei
;------------------------------
;       Position Counter
;               Position is incremented when going down and decremented when
;               going up.  The zero position is taken to be the upper edge of the pass
;               point signal (i.e. the falling edge in the up direction, the rising edge in
;               the down direction)
;------------------------------
        cp      STATE, #UP_DIRECTION               ; Test for the proper direction of the counter
        jr      z, DecPos                          ;
        cp      STATE, #STOP                       ;
        jr      z, DecPos                          ;
        cp      STATE, #UP_POSITION        ;
        jr      z, DecPos                          ;

IncPos:

        incw    POSITION                           ;
        cp      PPOINT_DEB, #2                     ; Test for pass point being seen
        jr      ult, NoDnPPoint                    ; If signal is low, none seen

DnPPoint:

        or      PassCounter, #10000000b            ; Mark pass point as currently high
        jr      CtrDone                            ;

NoDnPPoint:

        tm      PassCounter, #10000000b            ; Test for pass point seen before
        jr      z, PastDnEdge              ; If not, then we're past the edge
AtDnEdge:
        cp      L_A_C, #074H               ; Test for learning limits
        jr      nz, NormalDownEdge         ; if not, treat normally
LearnDownEdge:
        di
        sub     UP_LIMIT_LO, POSITION_LO   ; Set the up position higher
        sbc     UP_LIMIT_HI, POSITION_HI   ;
        dec     PassCounter                        ; Count pass point as being seen
        jr      Lowest1                            ; Clear the position counter
NormalDownEdge:
        dec     PassCounter                        ; Mark as one pass point closer to floor
        tm      PassCounter, #01111111b            ; Test for lowest pass point
        jr      nz, NotLowest1                     ; If not, don't zero the position counter
Lowest1:
        di
        clr     POSITION_HI                        ; Set the position counter back to zero
        ld      POSITION_LO, #1                    ;
        ei                                         ;
NotLowest1:
        cp      STATUS, #RSSTATUS                  ; Test for in RS232 mode
        jr      z, DontResetWall3                  ; If so, don't blink the LED
        ld      STATUS, #WALLOFF                   ; Blink the LED for pass point
        clr     VACFLASH                           ; Set the turn-off timer
DontResetWall3:
```

```
PastDnEdge:
NoUpPPoint:
        and     PassCounter, #01111111b           ; Clear the flag for pass point high
        jr      CtrDone                           ;

DecPos:

        decw    POSITION                          ;
        cp      PPOINT_DEB, #2                    ; Test for pass point being seen
        jr      ult, NoUpPPoint                   ; If signal is low, none seen

UpPPoint:

        tm      PassCounter, #10000000b           ; Test for pass point seen before
        jr      nz, PastUpEdge                    ; If so, then we're past the edge
AtUpEdge:
        tm      PassCounter, #01111111b           ; Test for lowest pass point
        jr      nz, NotLowest2                    ; If not, don't zero the position counter
Lowest2:
        di
        clr     POSITION_HI                       ; Set the position counter back to zero
        clr     POSITION_LO                       ;
        ei                                        ;
NotLowest2:
        cp      STATUS, #RSSTATUS                 ; Test for in RS232 mode
        jr      z, DontResetWall2                 ; If so, don't blink the LED
        ld      STATUS, #WALLOFF                  ; Blink the LED for pass point
        clr     VACFLASH                          ; Set the turn-off timer
DontResetWall2:
        inc     PassCounter                       ; Mark as one pass point higher above
        cp      PassCounter, FirstRun             ; Test for pass point above max. value
        jr      ule, PastUpEdge                   ; If not, we're fine
        ld      PassCounter, FirstRun             ; Otherwise, correct the pass counter
PastUpEdge:
        or      PassCounter, #10000000b           ; Set the flag for pass point high before

CtrDone:
RejectTheRPM:

        pop     rp                                ; return the rp
        iret                                      ; return


;------------------------------------------------------------------------
;       THIS IS THE SWITCH TEST SUBROUTINE
;
;       STATUS
;       0 => COMMAND TEST
;       1 => WORKLIGHT TEST
;       2 => VACATION TEST
;       3 => CHARGE
;       4 => RSSTATUS -- In RS232 mode, don't scan for switches
;       5 => WALLOFF  -- Turn off the wall control LED
;
;       SWITCH DATA
;       0 => OPEN
;       1 => COMMAND  CMD_SW
;       2 => WORKLIGHT       LIGHT_SW
;       4 => VACATION        VAC_SW
;------------------------------------------------------------------------


switches:

        ei
;4-22-97
        CP      LIGHT_DEB,#0FFH                   ;is the light button being held?
        JR      NZ,NotHeldDown                    ;if not debounced, skip long hold
        CP      EnableWorkLight,#01100000B ;has the 10 sec. already passed?
        JR      GE,HeldDown
        CP      EnableWorkLight,#01010000B
        JR      LT,HeldDown
        LD      EnableWorkLight,#10000000B ;when debounce occurs, set register
                                           ;to initiate e2 write in mainloop
        JR      HeldDown
NotHeldDown:
        CLR     EnableWorkLight
HeldDown:
;
;       and     SW_DATA, #LIGHT_SW          ; Clear all switches except for worklight
        cp      STATUS, #WALLOFF                    ; Test for illegal status
        jp      ugt, start                          ; if so reset
        jr      z, NoWallCtrl               ; Turn off wall control state
        cp      STATUS, #RSSTATUS                   ; Check for in RS232 mode
        jr      z, NOTFLASHED               ; If so, skip the state machine
        cp      STATUS,#3                           ; test for illegal number
        jp      z,charge                            ; if it is 3 then goto charge
        cp      STATUS,#2                           ; test for vacation
        jp      z,VACATION_TEST                     ; if so then jump
        cp      STATUS,#1                           ; test for worklight
        jp      z,WORKLIGHT_TEST                    ; if so then jump
                                                    ; else it id command
COMMAND_TEST:
        cp      VACFLAG,#00H                ; test for vacation mode
        jr      z,COMMAND_TEST1                     ; if not vacation skip flash

        inc     VACFLASH                            ; increase the vacation flash timer
        cp      VACFLASH,#10                ; test the vacation flash period
        jr      ult,COMMAND_TEST1                   ; if lower period skip flash
        and     p3,#~CHARGE_SW                      ; turn off wall switch
        or      p3,#DIS_SW                          ; enable discharge
        cp      VACFLASH,#60                ; test the time delay for max
        jr      nz,NOTFLASHED                       ; if the flash is not done jump and ret
        clr     VACFLASH                            ; restart the timer
NOTFLASHED:
        ret                                         ; return

NoWallCtrl:
        and     P3, #~CHARGE_SW                     ; Turn off the circuit
        or      P3, #DIS_SW                         ;
        inc     VACFLASH                            ; Update the off time
        cp      VACFLASH, #50               ; If off time hasn't expired,
        jr      ult, KeepOff                ; keep the LED off
        ld      STATUS, #CHARGE                     ; Reset the wall control
        ld      SWITCH_DELAY, #CMD_DEL_EX   ; Reset the charge timer
KeepOff:
        ret                                         ;

COMMAND_TEST1:
        tm      p0,#SWITCHES1                       ; command sw pressed?
        jr      nz,CMDOPEN                          ; open command
        tm      P0,#SWITCHES2                       ; test the second command input
        jr      nz,CMDOPEN
CMDCLOSED:                                          ; closed command
;       call    DECVAC                      ; decrease vacation debounce
;       call    DECLIGHT                            ; decrease light debounce
        cp      CMD_DEB,#0FFH                       ; test for the max number
        jr      z,SKIPCMDINC                ; if at the max skip inc
        di
        inc     CMD_DEB                             ; increase the debouncer
        inc     BCMD_DEB                            ; increase the debouncer
        ei
SKIPCMDINC:
        cp      CMD_DEB,#CMD_MAKE           ;
        jr      nz,CMDEXIT                          ; if not made then exit
        call    CmdSet                      ; Set the command switch
CMDEXIT:
      or    p3,#CHARGE_SW                     ; turn on the charge system
      and   p3,#~DIS_SW                       ;
      ld    SWITCH_DELAY,#CMD_DEL_EX    ; set the delay time to 8mS
      ld    STATUS,#CHARGE                    ; charge time
CMDDELEXIT:
      ret                                     ;

CmdSet:
      cp    L_A_C, #070H                ; Test for in learn limits mode
      jr    ult, RegCmdMake                   ; If not, treat as normal command
      jr    ugt, LeaveLAC               ; If learning, command button exits
      call  SET_UP_NOBLINK                    ; Set the up direction state
      jr    CMDMAKEDONE                       ;
RegCmdMake:
      cp    LEARNDB, #0FFH                    ; Test for learn button held
      jr    z, GoIntoLAC                ; If so, enter the learn mode
NormalCmd:
      di
      ld    LAST_CMD,#055H                    ; set the last command as command
cmd:  ld    SW_DATA,#CMD_SW                   ; set the switch data as command
      cp    AUXLEARNSW,#100                   ; test the time
      jr    ugt,SKIP_LEARN
      push  RP
      srp   #LEARNEE_GRP
      call  SETLEARN                          ; set the learn mode
      clr   SW_DATA                           ; clear the cmd
      pop   RP
      or    p0,#LIGHT_ON                ; turn on the light
      call  TURN_ON_LIGHT                     ; turn on the light
CMDMAKEDONE:
SKIP_LEARN:
      ld    CMD_DEB,#0FFH                     ; set the debouncer to ff one shot
      ld    BCMD_DEB,#0FFH                    ; set the debouncer to ff one shot
      ei
      ret

LeaveLAC:
      clr   L_A_C                             ; Exit the learn mode
      or    ledport,#ledh               ; turn off the LED for program mode
      call  SET_STOP_STATE                    ;
      jr    CMDMAKEDONE                       ;

GoIntoLAC:
      ld    L_A_C, #070H                ; Start the learn limits mode
      clr   FAULTCODE                         ; Clear any faults that exist
      clr   CodeFlag                          ; Clear the regular learn mode
      ld    LEARNT, #0FFH               ; Turn off the learn timer
      ld    ERASET, #0FFH               ; Turn off the erase timer
      jr    CMDMAKEDONE                       ;


CMDOPEN:                                      ; command switch open
      and   p3,#~CHARGE_SW                    ; turn off charging sw
      or    p3,#DIS_SW                        ; enable discharge
      ld    DELAYC,#16                        ; set the time delay
DELLOOP:
      dec   DELAYC
      jr    nz,DELLOOP                        ; loop till delay is up
      tm    p0,#SWITCHES1                     ; command line still high
      jr    nz,TESTWL                         ; if so return later
      call  DECVAC                      ; if not open line dec all debouncers
      call  DECLIGHT                          ;
      call  DECCMD                      ;
      ld    AUXLEARNSW,#0FFH                  ; turn off the aux learn switch
      jr    CMDEXIT                           ; and exit

TESTWL:
      ld    STATUS,#WL_TEST                   ; set to test for a worklight
      ret                                     ; return
```

```
WORKLIGHT_TEST:
        tm      p0,#SWITCHES1                           ; command line still high
        jr      nz,TESTVAC2                             ; exit setting to test for vacation
        call    DECVAC                          ; decrease the vacation debouncer
        call    DECCMD                          ; and the command debouncer
        cp      LIGHT_DEB,#0FFH                         ; test for the max
        jr      z,SKIPLIGHTINC                          ; if at the max skip inc
        inc     LIGHT_DEB                               ; inc debouncer
SKIPLIGHTINC:
        cp      LIGHT_DEB,#LIGHT_MAKE                   ; test for the light make
        jr      nz,CMDEXIT                              ; if not then recharge delay
        call    LightSet                                ; Set the light debouncer
        jr      CMDEXIT                                 ; then recharge

LightSet:
        ld      LIGHT_DEB,#0FFH                         ; set the debouncer to max
        ld      SW_DATA,#LIGHT_SW               ; set the data as worklight
        cp      RRTO,#RDROPTIME                         ; test for code reception
        jr      ugt,CMDEXIT                             ; if not then skip the seting of flag
        clr     AUXLEARNSW                              ; start the learn timer
        ret

TESTVAC2:
        ld      STATUS,#VAC_TEST                        ; set the next test as vacation
        ld      switch_delay,#VAC_DEL                   ; set the delay
LIGHTDELEXIT:
        ret                                             ; return

VACATION_TEST:
        djnz    switch_delay,VACDELEXIT                 ;

        tm      p0,#SWITCHES1                           ; command line still high
        jr      nz,EXIT_ERROR                           ; exit with a error setting open state
;       call    DECLIGHT                                ; decrease the light debouncer
        call    DECCMD                          ; decrease the command debouncer
        cp      VAC_DEB,#0FFH                           ; test for the max
        jr      z,VACINCSKIP                            ; skip the incrementing
        inc     VAC_DEB                                 ; inc vacation debouncer
VACINCSKIP:
        cp      VACFLAG,#00H                    ; test for vacation mode
        jr      z,VACOUT                                ; if not vacation use out time
VACIN:
        cp      VAC_DEB,#VAC_MAKE_IN                    ; test for the vacation make point
        jr      nz,VACATION_EXIT                        ; exit if not made
        call    VacSet                          ;
        jr      VACATION_EXIT                           ;

VACOUT:
        cp      VAC_DEB,#VAC_MAKE_OUT                   ; test for the vacation make point
        jr      nz,VACATION_EXIT                        ; exit if not made
        call    VacSet                          ;
        jr      VACATION_EXIT                   ; Forget vacation mode

VacSet:
        ld      VAC_DEB,#0FFH                           ; set vacation debouncer to max
        cp      AUXLEARNSW,#100                         ; test the time
        jr      ugt,SKIP_LEARNV
        push    RP
        srp     #LEARNEE_GRP
        call    SETLEARN                                ; set the learn mode
        pop     RP
        or      p0, #LIGHT_ON                   ; Turn on the worklight
        call    TURN_ON_LIGHT                   ;
        ret

SKIP_LEARNV:
        ld      VACCHANGE,#0AAH                         ; set the toggle data
        cp      RRTO,#RDROPTIME                 ; test for code reception
        jr      ugt,VACATION_EXIT               ; if not then skip the seting of flag
        clr     AUXLEARNSW                      ; start the learn timer
VACATION_EXIT:
        ld      SWITCH_DELAY,#VAC_DEL_EX   ; set the delay
        ld      STATUS,#CHARGE                  ; set the next test as charge
VACDELEXIT:
        ret

EXIT_ERROR:
        call    DECCMD                     ; decrement the debouncers
        call    DECVAC                     ;
        call    DECLIGHT                        ;
        ld      SWITCH_DELAY,#VAC_DEL_EX   ; set the delay
        ld      STATUS,#CHARGE                  ; set the next test as charge
        ret


charge:
        or      p3,#CHARGE_SW                   ;
        and     p3,#~DIS_SW                     ;
        dec     SWITCH_DELAY               ;
        jr      nz,charge_ret                   ;
        ld      STATUS,#CMD_TEST                ;
charge_ret:
        ret


DECCMD:
        cp      CMD_DEB,#00H               ; test for the min number
        jr      z,SKIPCMDDEC               ; if at the min skip dec
        di
        dec     CMD_DEB                         ; decrement debouncer
        dec     BCMD_DEB                        ; decrement debouncer
        ei
SKIPCMDDEC:
        cp      CMD_DEB,#CMD_BREAK         ; if not at break then exit
        jr      nz,DECCMDEXIT                   ; if not break then exit
        call    CmdRel                     ;
DECCMDEXIT:
        ret                                     ; and exit

CmdRel:
        cp      L_A_C, #070H               ; Test for in learn mode
        jr      nz, NormCmdBreak                ; If not, treat normally
        call    SET_STOP_STATE                  ; Stop the door
NormCmdBreak:
        di
        clr     CMD_DEB                         ; reset the debouncer
        clr     BCMD_DEB                        ; reset the debouncer
        ei
        ret


DECLIGHT:
        cp      LIGHT_DEB,#00H                  ; test for the min number
        jr      z,SKIPLIGHTDEC                  ; if at the min skip dec
        dec     LIGHT_DEB                       ; decrement debouncer
SKIPLIGHTDEC:
        cp      LIGHT_DEB,#LIGHT_BREAK          ; if not at break then exit
        jr      nz,DECLIGHTEXIT                 ; if not break then exit
        clr     LIGHT_DEB                       ; reset the debouncer
DECLIGHTEXIT:
        ret                                     ; and exit


DECVAC:
        cp      VAC_DEB,#00H               ; test for the min number
        jr      z,SKIPVACDEC                ; if at the min skip dec
        dec     VAC_DEB                             ; decrement debouncer
SKIPVACDEC:
        cp      VACFLAG,#00H                ; test for vacation mode
        jr      z,DECVACOUT                         ; if not vacation use out time
DECVACIN:
        cp      VAC_DEB,#VAC_BREAK_IN               ; test for the vacation break point
        jr      nz,DECVACEXIT                       ; exit if not
        jr      CLEARVACDEB                 ;

DECVACOUT:
        cp      VAC_DEB,#VAC_BREAK_OUT              ; test for the vacation break point
        jr      nz,DECVACEXIT                       ; exit if not
CLEARVACDEB:
        clr     VAC_DEB                             ; reset the debouncer
DECVACEXIT:
        ret                                         ; and exit


;-----------------------------------------------------------------------
;       FORCE TABLE
;-----------------------------------------------------------------------
force_table:

f_0     .byte  000H,  06BH,  06CH
        .byte  000H,  06BH,  06CH
        .byte  000H,  06DH,  073H
        .byte  000H,  06FH,  08EH
        .byte  000H,  071H,  0BEH
        .byte  000H,  074H,  004H
        .byte  000H,  076H,  062H
        .byte  000H,  078H,  0DAH
        .byte  000H,  07BH,  06CH
        .byte  000H,  07EH,  01BH
        .byte  000H,  080H,  0E8H
        .byte  000H,  083H,  0D6H
        .byte  000H,  086H,  09BH
        .byte  000H,  089H,  07FH
        .byte  000H,  08CH,  084H
        .byte  000H,  08FH,  0ABH
        .byte  000H,  092H,  0F7H
        .byte  000H,  096H,  06BH
        .byte  000H,  09AH,  009H
        .byte  000H,  09DH,  0D5H
        .byte  000H,  0A1H,  0D2H
        .byte  000H,  0A6H,  004H
        .byte  000H,  0AAH,  076H
        .byte  000H,  0AFH,  027H
        .byte  000H,  0B4H,  01CH
        .byte  000H,  0B9H,  05BH
        .byte  000H,  0BEH,  0EBH
        .byte  000H,  0C4H,  0D3H
        .byte  000H,  0CBH,  01BH
        .byte  000H,  0D1H,  0CDH
        .byte  000H,  0D8H,  0F4H
        .byte  000H,  0E0H,  09CH
        .byte  000H,  0E7H,  01CH
        .byte  000H,  0EDH,  0FFH
        .byte  000H,  0F5H,  04FH
        .byte  000H,  0FDH,  015H
        .byte  001H,  005H,  05DH
        .byte  001H,  00EH,  035H
        .byte  001H,  017H,  0ABH
        .byte  001H,  021H,  0D2H
        .byte  001H,  02CH,  0BBH
        .byte  001H,  038H,  0E0H
        .byte  001H,  045H,  03AH
        .byte  001H,  053H,  008H
        .byte  001H,  062H,  010H
```

```
      .byte  001H,  072H,  07DH
      .byte  001H,  084H,  083H
      .byte  001H,  098H,  061H
      .byte  001H,  0AEH,  064H
      .byte  001H,  0C6H,  0E8H
      .byte  001H,  0E2H,  062H
      .byte  002H,  001H,  065H
      .byte  002H,  024H,  0AAH
      .byte  002H,  04DH,  024H
      .byte  002H,  07CH,  010H
      .byte  002H,  0B3H,  01EH
      .byte  002H,  0F4H,  094H
      .byte  003H,  043H,  0C1H
      .byte  003H,  0A5H,  071H
      .byte  004H,  020H,  0FCH
      .byte  004H,  0C2H,  038H
      .byte  005H,  09DH,  080H
      .byte  013H,  012H,  0D0H
f_63: .byte  013H,  012H,  0D0H

SIM_TABLE:

              .WORD  00000H        ; Numbers set to zero (proprietary table)
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H
              .WORD  00000H

SPEED_TABLE_50:

      .BYTE  40
      .BYTE  34
      .BYTE  32
      .BYTE  30
      .BYTE  28
      .BYTE  27
      .BYTE  25
      .BYTE  24
      .BYTE  23
      .BYTE  21
      .BYTE  20
      .BYTE  19
      .BYTE  17
      .BYTE  16
      .BYTE  15
      .BYTE  13
      .BYTE  12
      .BYTE  10
      .BYTE  8
      .BYTE  6
      .BYTE  0

SPEED_TABLE_60:

      .BYTE  33
      .BYTE  29
      .BYTE  27
      .BYTE  25
```

```
  .BYTE  23
  .BYTE  22
  .BYTE  21
  .BYTE  20
  .BYTE  19
  .BYTE  18
  .BYTE  17
  .BYTE  16
  .BYTE  15
  .BYTE  13
  .BYTE  12
  .BYTE  11
  .BYTE  10
  .BYTE  8
  .BYTE  7
  .BYTE  5
  .BYTE  0

  ; Fill 49 bytes of unused memory

  FILL10
  FILL10
  FILL10
  FILL10
  FILL
  FILL
  FILL
  FILL
  FILL
  FILL
  FILL
  FILL
  FILL

.end
```